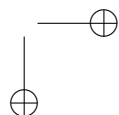
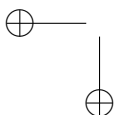
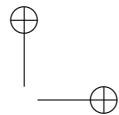
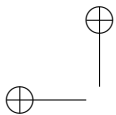




Roma Tre University  
Ph.D. in Computer Science and Engineering

# Outsourced Storage Services: Authentication and Security Visualization

Bernardo Palazzi



# Outsourced Storage Services: Authentication and Security Visualization

A thesis presented by  
Bernardo Palazzi  
in partial fulfillment of the requirements for the degree of  
Doctor of Philosophy  
in Computer Science and Engineering  
Roma Tre University  
Dept. of Informatics and Automation  
March 2009

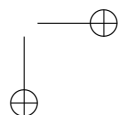
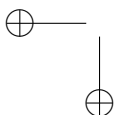
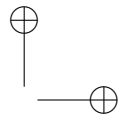
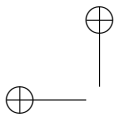
COMMITTEE:

*Prof. Giuseppe Di Battista*

REVIEWERS:

*Prof. Pierangela Samarati*

*Prof. Roberto Tamassia*



## Abstract

We address the problem of authenticating data in outsourced, often untrusted, services, when a user stores more or less confidential information in a remote service such as an online calendar, remote storage, outsourced DBMS, and others. How can outsourced data be proven authentic?

Data authentication captures the security needs of many computing applications that save and use sensitive information in hostile remote distributed environments and its importance increases, given the current trend in modern system design towards outsourced services with minimal trust assumptions. Solutions should not only be provably secure, but efficient and easily implementable.

This dissertation presents an extensive study of data authentication and introduces a general method, based on a security *middleware*, external to the service, that performs authentication operations in parallel with standard service functions to minimize the time overhead. We examine the problem for different services, and design efficient new techniques with authenticating general classes of operations, such as relational primitives, multidimensional queries and *relational join* and remote storage management.

Another important issue that we cover in this dissertation is the security usability of outsourced services. In particular we analyze the information security visualization techniques and we address the problem of file permissions visualization. *TrACE*, a prototype tool based on a *treemap* is presented with an extensive user study to show the usability improvement of this tool.

## Acknowledgments

I would like to thank my Ph.D. advisor Professor Giuseppe Di Battista for his patient guidance and training. He taught me how to do research in computer security, inspired me to think *out of the box*, and showed me how to keep always a different point of view in my work. He has also given me much valuable advice and considerate support for my personal life and my future career. Most importantly, Pino taught me to keep an open mind so that I could be able to tackle a broad spectrum of research problems.

Another person who influenced me most during my Ph.D. course is Professor Roberto Tamassia. He taught me the fundamentals of modern authentication techniques, which is the foundation of my thesis work. Roberto also showed me how to write an interesting research paper and give a more interesting presentation.

I also want to thank Professor Pierangela Samarati for her extensive and intelligent review on my thesis work.

I feel honored to have been supported by ISCOM as a Ph.D. student and I wish to thank the Institute of Communication of the Italian Ministry of Economic Development - Communication and in particular its former Director Luisa Franchina Ph.D. for some useful discussions.

I would also like to thank Professors Maurizio Patrignani and Maurizio Pizzonia for working with me on various security problems. I learned tremendously from our collaboration. I also want to thank my Italian and American friends and colleagues: Alex, Babis, Claudio, Fabio, Fabrizio, Francesca, Gabriella, Giulia, Luca, Max, Nikos, Patrizio, Pier Francesco, Stefano and Tiziana for making my graduate school life enjoyable and memorable.

I would like to thank my very special *English teacher*, Laura, for her very kind help during all my period of stays in Providence.

In particular I would like to thank Norbert, my best friend, for his endless availability and support, he is the one that continues to have always the *right*

viii

English words.

Finally, I want to thank my family and in particular my parents, Marina and Tonino, my brother Tommaso, my aunt Stefania and my uncle Armando for their patience with me during my graduate studies. This thesis is dedicated to my grandmothers, Nora and Olimpia, although, unfortunately, they are no more among us.

# Contents

<b>Contents</b>	<b>ix</b>
<b>1 Introduction</b>	<b>1</b>
1.1 Outsourced Storage Authentication . . . . .	3
1.2 Overview of Authenticated Data Structures . . . . .	4
1.3 Authenticated Skip List . . . . .	7
1.4 Overview and Thesis Structure . . . . .	12
<b>2 Authenticated Relational Tables and Authenticated Skip Lists</b>	<b>15</b>
2.1 Introduction . . . . .	15
2.2 The Reference Model . . . . .	16
2.3 A Fine Grained Approach . . . . .	21
2.4 Exploiting Nested Sets . . . . .	22
2.5 Experimental Evaluation . . . . .	24
2.6 Conclusions . . . . .	27
<b>3 Multi-Column Authentication</b>	<b>29</b>
3.1 Introduction . . . . .	29
3.2 The Authentication Problem . . . . .	31
3.3 Architecture . . . . .	36
3.4 Our Approach . . . . .	38
3.5 Experimental Evaluation . . . . .	47
3.6 Conclusions . . . . .	53
<b>4 Network Storage Integrity</b>	<b>55</b>
4.1 Introduction . . . . .	55
4.2 Our Approach . . . . .	57
4.3 Implementation . . . . .	67

4.4	Experiments . . . . .	72
4.5	Conclusions . . . . .	78
<b>5</b>	<b>Graph Drawing for Security Visualization</b>	<b>79</b>
5.1	Introduction . . . . .	79
5.2	Network Monitoring . . . . .	81
5.3	Border Gateway Protocol . . . . .	88
5.4	Access Control . . . . .	92
5.5	Trust Negotiation . . . . .	94
5.6	Attack Graphs . . . . .	94
5.7	Conclusions . . . . .	95
<b>6</b>	<b>Access-Control Visualization</b>	<b>97</b>
6.1	Introduction . . . . .	97
6.2	Preliminaries . . . . .	99
6.3	Effective Access Control Visualization . . . . .	100
6.4	The TrACE (Treemap Access Control Evaluator) Tool. . . . .	103
6.5	User Feedback . . . . .	106
6.6	Conclusions . . . . .	117
	<b>Conclusion</b>	<b>119</b>
	Summary of Results . . . . .	119
	Future Directions . . . . .	120
	<b>Appendices</b>	<b>121</b>
	<b>TrACE User Study</b>	<b>123</b>
	<b>Bibliography</b>	<b>129</b>

## Chapter 1

# Introduction

This dissertation addresses the problem of authentication data that is retrieved through an outsourced storage service: when the repository of the data is not managed directly by the end-user, and the manager of data is not completely trusted, how can data received be proven authentic? This question is the core of several security-related problems underlying any real-life computing application that involves storage of data over a communication or computing structure that can act unreliably. Clearly, data authentication ensuring that received data can be accurately verified to be in its original form is a fundamental problem in the area of information security, for information is valuable only when it is trustworthy.

Data authentication captures some primary security needs of today’s computing reality. In fact, integrity checking of data and data structures has grown in importance recently due to the expansion of online services, which have become reliable and scalable, and often have a pay-per-use cost model with affordable rates.

The architecture analyzed is based on a client-server model see Fig. 1.1. The client corresponds to a minimal trusted system component and the server corresponds to an untrusted system device. Stored data is organized according to storage services of generic form (i.e. a database, a file server, etc.), which are managed remotely by the client, without the need to use secure communication. The client constitutes an authentication module that verifies the correctness of both incoming and outgoing data: any update operation is effectively enforced to perform correctly and any query operation is effectively verified to return valid data. This module achieves verification by keeping minimal system

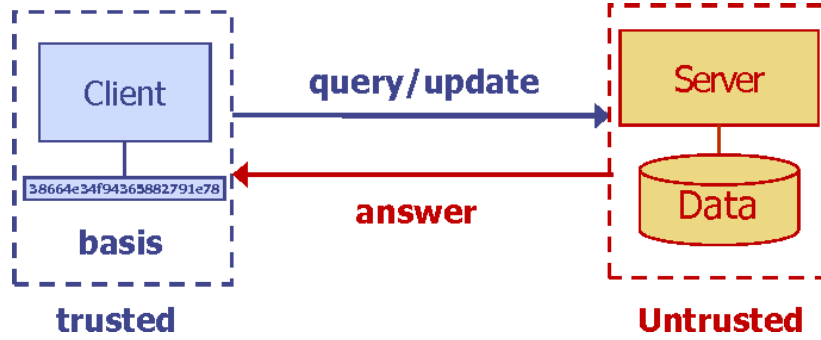


Figure 1.1: The client does not trust the server and verifies each answer (result of an operation) returned by the server.

state and processing small amounts of information provided by a certification module at the server, the latter involving minimal or no processing. Thus, the proposed approach realizes a transparent security layer for authenticated outsourced storage.

This dissertation presents an extensive study on the data authentication problem. We examine the problem for structured data that is dynamically maintained in outsourced storage services. We provide formal problem definitions that carefully model the notions of security and efficiency in data authentication, and design new efficient techniques for securely authenticating general classes of query problems, such as queries on multidimensional relational databases, and integrity checking of untrusted network storage. We also study the complexity of the problem and the computational and communication costs that are inherently associated with the authentication of data. Moreover, in a general computational and data querying model, we provide a new framework for authenticating any query type over structured and dynamic data. By decoupling the answer-generation and answer-validation procedures, this framework can exploit a strong parallelism, where an authenticated query result can be validated without time overhead if the data result is bigger than a few dozen bytes. Finally, we address the problem of security usability through visualization techniques. We start with a survey on graph drawing techniques for security visualization and we conclude with the application of this technique to introduce a new approach to visualize storage access control policy.

## 1.1 The problem of Outsourced Storage Authentication

Corporations and consumers increasingly trust their data to outsourced resources and want to be assured that no one alters or deletes it. Commercial network storage applications are rapidly growing, with services that range from general file storage to web operating systems. Outsourced storage systems sometimes also offer services to assure confidentiality (through encryption) and integrity of data transmission (typically through checksum hashes). However, they do not provide a solution to the storage integrity problem.

Thus, the client would have to develop its own authentication solution, such as a cache of the hashes of the data items, in order to verify that data returned by the storage server has not been tampered with. In the remainder of this dissertation, we use the term “authentication” to refer to the verification of the authenticity, or integrity, of *data*, as opposed to user identity authentication, which is a separate security issue.

It is sometimes assumed that symmetric encryption may be a solution for multiple security problems, but in fact, integrity checking and encryption are orthogonal services (see, e.g., [57]). For example, if we only encrypt files, an attacker can remove some files without our knowledge since decryption will still work perfectly on the remaining files. Only an integrity checking service can detect such an attack.

To deal with these problems, we propose a simple architecture that consists of three main parties:

- The *storage server* stores some outsourced data. The storage server is untrusted and can be any storage service available online.
- The *authentication server* stores and processes authentication information of the outsourced data. The authentication server is also untrusted and can be an outsourced computational resource.
- The *client* queries and updates both the storage server and the authentication server and verifies the results returned by them. We assume that no one can interfere with the state, computation and storage at the client. Of course, it is possible in the real world for a client to be compromised, but we are only interested in protecting the client against errors and malicious behavior by the storage server and authentication server.

## 1.2 Overview of Authenticated Data Structures

Throughout this section, we denote with  $n$  the size of the collection  $S$  maintained by an authenticated data structure.

Early work on authenticated data structures was motivated by the *certificate revocation* problem in public key infrastructure and focused on *authenticated dictionaries*, on which membership queries are performed.

The *hash tree* scheme introduced by Merkle [64, 66] can be used to implement a static authenticated dictionary. A hash tree  $T$  for a set  $S$  stores cryptographic hashes of the elements of  $S$  at the leaves of  $T$  and a value at each internal node, which is the result of computing a cryptographic hash function on the values of its children. The hash tree uses linear space and has  $O(\log n)$  proof size, query time and verification time. A dynamic authenticated dictionary based on hash trees that achieves  $O(\log n)$  update time is described in [74]. A dynamic authenticated dictionary that uses a hierarchical hashing technique over skip lists is presented in [36]. This data structure also achieves  $O(\log n)$  proof size, query time, update time and verification time. Other schemes based on variations of hash trees have been proposed in [13, 31, 50].

A detailed analysis of the efficiency of authenticated dictionary schemes based on hierarchical cryptographic hashing is conducted in [99], where precise measures of the computational overhead due to authentication are introduced. Using this model, lower bounds on the authentication cost are given, existing authentication schemes are analyzed and a new authentication scheme is presented that achieve performance very close to the theoretical optimal.

An alternative approach to the design of authenticated dictionary, based on the *RSA accumulator*, is presented in [40]. This technique achieves constant proof size and verification time and provides a tradeoff between the query and update times. For example, one can achieve  $O(\sqrt{n})$  query time and update time.

In [3], the notion of a *persistent authenticated dictionary* is introduced, where the user can issue historical queries of the type “was element  $e$  in set  $S$  at time  $t$ ”.

A first step towards the design of more general authenticated data structures (beyond dictionaries) is made in [22] with the authentication of relational database operations and multidimensional orthogonal range queries.

In [62], a general method for designing authenticated data structures using hierarchical hashing over a search graph is presented. This technique is applied to the design of static authenticated data structures for pattern matching in tries and for orthogonal range searching in a multidimensional set of points.

## 1.2. OVERVIEW OF AUTHENTICATED DATA STRUCTURES

5

Efficient authenticated data structures supporting a variety of fundamental search problems on graphs (e.g., path queries and biconnectivity queries) and geometric objects (e.g., point location queries and segment intersection queries) are presented in [42]. This paper also provides a general technique for authenticating data structures that follow the *fractional cascading* paradigm.

The software architecture and implementation of an authenticated dictionary based on skip lists is presented in [41]. A distributed system realizing an authenticated dictionary, is described in [37]. This paper also provides an empirical analysis of the performance of the system in various deployment scenarios. The authentication of distributed data using web services and XML signatures is investigated in [85]. *Prooflets*, a scalable architecture for authenticating web content based on authenticated dictionaries, are introduced in [94].

Work related to authenticated data structures includes [14, 20, 38, 58, 59]

In particular there has been a considerable amount of work done on untrusted outsourced storage. On problems concerning confidentiality and privacy preservation, through encryption, in outsourced systems are discussed in [24, 25] these topics are orthogonal and complementary to our work.

Yumerefendi and Chase [107] propose a solution for authenticated network storage, using a Merkle tree [66] as the underlying data structure. PKI is used, however, and the basis (a trusted hash value associated with an authenticated data structure — see Section 1.3) is outsourced to an external medium, raising communication and security issues. Oprea and Reiter [81] present a solution for authenticated storage of files that takes advantage of the entropy of individual blocks. The client keeps hash values only for high-entropy blocks that pass a randomness test. A solution for authenticating an outsourced file system (hierarchically organized) is presented by Jammalamadaka et al. [46]. However their processing of updates is computationally expensive. Fu et al. [30] describe and implement a method for efficiently and securely accessing a read-only file system that has been distributed to many providers. The Athos architecture, developed by Goodrich et al. [35], is a solution for efficiently authenticating operations on an outsourced file system that is related to our approach. The system that we use and Athos both leverage algorithms described by Papamantou and Tamassia [84] for querying and updating two-party authenticated data structures.

Untrusted storage where one digital signature for each object is kept is presented by Goh et al. [33]. The SUNDR system, introduced by Mazières et al. [55], protects data integrity in a fully distributed setting by digitally signing every operation and maintaining hash trees. The system requires off-line user collaboration for protection against replay attacks. Goodrich et al. [38] explore

data integrity for multi-authored dictionaries, where clients can efficiently validate a sequence of updates. A number of works focus on proving retrievability of outsourced data. Schwarz and Miller [91] propose a scheme that makes use of algebraic signatures to verify that data in a distributed system, safeguarded using erasure coding, is stored correctly. Shacham and Waters [93] give provably secure schemes for verifying retrievability that use homomorphic authenticators based on signatures. The model of *provable data possession* (PDP) is proposed by Ateniese et al. [5]. The authors specifically target systems storing very large amounts of data. The client keeps a constant-size digest of the data and the server can demonstrate the possession of a file or a block by returning a compact proof of possession. SafeStore, a system devised by Kotla et al. [52], combines redundancy and hierarchical erasure coding with auditing protocols for checking retrievability. A method for the authentication of outsourced databases using a signature scheme appears in papers by Mykletun et al. [72] and Narasimha and Tsudik [75]. In this approach, the client’s computation is computationally expensive. Also, the client has to engage in a multi-round protocol in order to perform an update.

Buldas, in [12], studies how to extend *ADS* to perform more complex queries and uses optimizations on interval queries. In [82, 83] the authors propose a method to authenticate projection queries using different cryptographic techniques for verifying the completeness of relational queries. While the papers are quite promising in terms of theoretical bounds and analysis, the practical efficiency is not demonstrated. Di Battista and Palazzi [23] present a method for outsourcing a dictionary, where a skip list is stored by the server into a table of a relational database management system (DBMS) and the client issues SQL queries to the DBMS to retrieve authentication information. Note that this method is fully applicable to our framework since the update of the basis is done at the client’s side, whenever an update occurs. A related solution is presented by Miklau and Suciu [67], where they proposed to embed into a relational table an *MHT*. However, the technique is described only partially and seems to have some drawbacks. Namely, validating the result of a query seems to require several distinct queries on the DBMS. This is in contrast with the typical atomicity requirements of concurrency. Also, the *MHT*s require frequent rebalancing for supporting updates and it is unclear how to match this requirement with the need to have a few updates in the relational table. Further, the time performance illustrated in the paper are not supported by a clear description of the experimental platform and show some inconsistency. For example in one of the tests the time requested for authentication decreases with the growth of the table.

### 1.3. AUTHENTICATED SKIP LIST

7

Maheshwari et al. [56] take a different approach to the authentication of a database, detailing a new trusted database (TDB) system with built-in support for integrity checking and encryption, and a performance advantage over architectures that add a layer of cryptography on top of a typical unsecured database. A survey for secure distributed storage is presented by Kher and Kim [49]. The archival storage of signed documents is studied by Maniatis and Baker [58].

An authenticated data structure three-party model, where the data owner outsources the data to a server, which answers queries issued by clients on behalf of the data owner. See [98] for a survey. A solution for the authentication of outsourced databases in the three-party model, using an authenticated B-tree for the indices, is presented by Li et al. [53]. Lower bounds on the client storage in the three-party model are given by Tamassia and Triandopoulos [100]. A method for authentication of XML documents is provided by Devanbu et al. [21].

### 1.3 Authenticated Skip List

For the purposes of this dissertation we need to provide a description of the skip lists. The skip list data structure [87] is an efficient tool for storing an ordered set of *elements*. It supports the following operations on a set of elements.

- **find**( $x$ ): Determine whether element  $x$  is in the set.
- **insert**( $x$ ): Insert element  $x$  into the set.
- **delete**( $x$ ): Remove element  $x$  from the set.

A skip list  $S$  stores a set of elements in a sequence of linked lists  $S_0, S_1, \dots, S_t$  called *levels*. The members of the lists are called *nodes*. The base list,  $S_0$ , stores in its nodes all the elements of  $S$  in order, as well as *sentinels* associated with the special elements  $-\infty$  and  $+\infty$ . Each list  $S_{i+1}$  stores a subset of the elements of  $S_i$ . The method used to define the subset from one level to the next determines the type of skip list. The default method is simply to choose the elements of  $S_{i+1}$  at random among the elements of  $S_i$  with probability  $\frac{1}{2}$ . One could also define a deterministic skip list [71], which uses simple rules to guarantee that between any two elements in  $S_i$  there are at least 1 and at most 3 elements of  $S_{i+1}$ . In either case, the sentinel elements  $-\infty$  and  $+\infty$  are always included in the next level up, and the top level, is maintained to be  $O(\log n)$ . We distinguish the node of the top list  $S_t$  storing  $-\infty$  as the start node  $s$ .

An element that is in  $S_{i-1}$  but not in  $S_i$  is said to be a *plateau element* of  $S_{i-1}$ . An element that is in both  $S_{i-1}$  and  $S_i$  is said to be a *tower element* in  $S_{i-1}$ . Thus, between any two tower elements, there are some plateau elements. In randomized skip lists, the expected number of plateau elements between two tower elements is one. The skip list of Fig. 1.2 has 7 elements (including sentinels). The element 6 is stored in 3 nodes with different level. The overall number of nodes is 17.

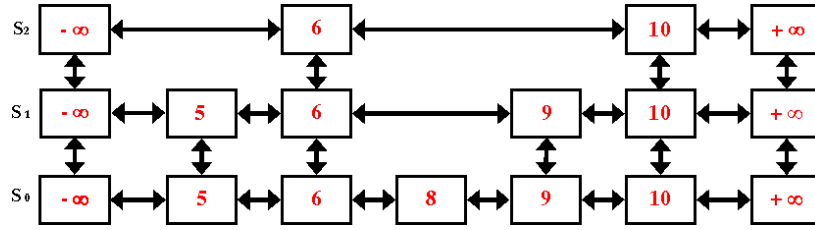


Figure 1.2: Skip List

To perform a search for element  $x$  in a skip list, we begin at the start node  $s$ . Let  $v$  denote the current node in our search (initially,  $v = s$ ). The search proceeds using two actions, *hop forward* and *drop down*, which are repeated one after the other until we terminate the search. See Fig. 1.3.

- *Hopforward*: We move right along the current list until we find the node of the current list with largest element less than or equal to  $x$ . That is, while  $elem(right(v)) < x$ , we perform  $v = right(v)$ .
- *Dropdown*: If  $down(v) = null$ , then we are done with our search: node  $v$  stores the largest element in the skip list less than or equal to  $x$ . Otherwise, we update  $v = down(v)$ .

In a deterministic skip list, the above searching process is guaranteed to take  $O(\log n)$  time. Even in a randomized skip list, it is fairly straightforward to show (e.g., see [39]) that the above searching process runs in expected  $O(\log n)$  time, for, with high probability, the height  $t$  of the randomized skip list is  $O(\log n)$  and the expected number of nodes visited on any level is 3.

To insert a new element  $x$ , we determine which lists should contain the new element  $x$  by a sequence of simulated random coin flips. Starting with  $i = 0$ , while the coin comes up heads, we use the stack  $A$  to trace our way back to

### 1.3. AUTHENTICATED SKIP LIST

9

the position of list  $S_{i+1}$  where element  $x$  should go, add a new node storing  $x$  to this list, and set  $i = i + 1$ . We continue this insertion process until the coin comes up tails. If we reach the top level with this insertion process, we add a new top level on top of the current one. The time taken by the above insertion method is  $O(\log n)$  with high probability. To delete an existing element  $x$ , we remove all the nodes that contain the element  $x$ . This takes time is  $O(\log n)$  with high probability.

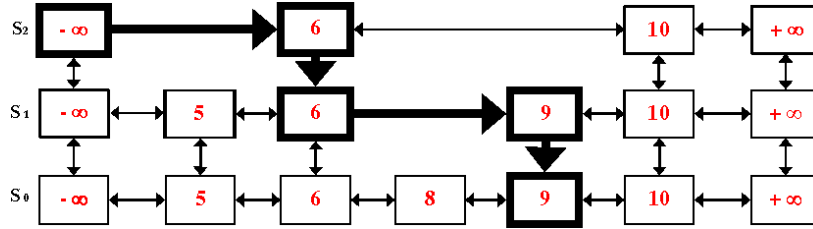


Figure 1.3: A value searching in a Skip List: search for element 9 in the skip list of Figure 1.2. The nodes visited and the links traversed are drawn with thick lines and arrows.

To introduce the *Authenticated Skip Lists* we need to use the commutative hash technique [34] developed by Goodrich and Tamassia. A hash function  $h$  is commutative if  $h(x; y) = h(y; x)$ , for all  $x$  and  $y$ . Given a cryptographic hash function  $h$  that is collision resistant in the usual sense, we construct a candidate commutative cryptographic hash function,  $h_0$ , as follows [34] :

$$h_0(x, y) = h(\min(x, y), \max(x, y))$$

It can be shown that  $h_0$  is commutatively collision resistant [34].

The authenticated skip list introduced in [34] consists of a skip list where each node  $v$  stores a label computed accumulating the elements of the set with a commutatively cryptographic hash function  $h$ . For completeness, let us review how hashing occurs. See [34] for details. For each node  $v$  we define label  $f(v)$  in terms of the respective values at nodes  $w = \text{right}(v)$  and  $u = \text{down}(v)$ . If  $\text{right}(v) = \text{null}$ , then we define  $f(v) = 0$ . The definition of  $f(v)$  in the general case depends on whether  $u$  exists or not for this node  $v$ .

- $u = \text{null}$ , i.e.,  $v$  is on the base level:
  - If  $w$  is a tower node, then
 
$$f(v) = h(\text{elem}(v), \text{elem}(w))$$

- If  $w$  is a plateau node, then  
 $f(v) = h(elem(v), f(w))$ .
- $u \neq null$ , i.e.,  $v$  is not on the base level:
  - If  $w$  is a tower node, then  
 $f(v) = f(u)$ .
  - If  $w$  is a plateau node, then  
 $f(v) = h(f(u), f(w))$ .

We illustrate the flow of the computation of the hash values labeling the nodes of a skip list in See Fig. 1.4. Note that the computation flow defines a directed acyclic graph *DAG*, not a tree. After performing the update in the skip list, the hash values must be updated to reflect the change that has occurred. The additional computational expense needed to update all these values is expected with high probability to be  $O(\log n)$ . The verification of the answer to a query is simple, thanks to the use of a commutative hash function. Recall that the goal is to produce a verification that some element  $x$  is or is not contained in the skip list. In the case when the answer is “yes”, we verify the presence of the element itself. Otherwise, we verify the presence of two elements  $x_a$  and  $x_b$  stored at consecutive nodes on the bottom level  $S_0$  such that  $x_a < x < x_b$ . In either case, the answer authentication information is a single sequence of values, together with the signed, timestamped, label  $f(s)$  of the start node  $s$ .

Let  $P(x) = (v_1; \dots; v_m)$  be the sequence of nodes that are visited when searching for element  $x$ , in reverse order. In the example of Fig. 1.5, we have  $P(9)$  that needs not only the nodes  $(9, 6, -\infty)$  with the thick line but also all the siblings with the stroke dash-dot-dash-dot. Note that by the properties of a skip list, the size  $m$  of sequence  $P(x)$  is  $O(\log n)$  with high probability. We construct from the node sequence  $P(x)$  a sequence  $Q(x) = (y_1; \dots; y_m)$  of values such that:

- $y_m = f(s)$ , the label of the start node;
- $y_m = h(y_{m-1}; h(y_{m-2}; h(\dots; y_1) \dots))$

The user verifies the answer for element  $x$  by simply hashing the values of the sequence  $P(x)$  in the given order, and comparing the result with the signed value  $f(s)$ , where  $s$  is the start node of the skip list. If the two values agree,

### 1.3. AUTHENTICATED SKIP LIST

11

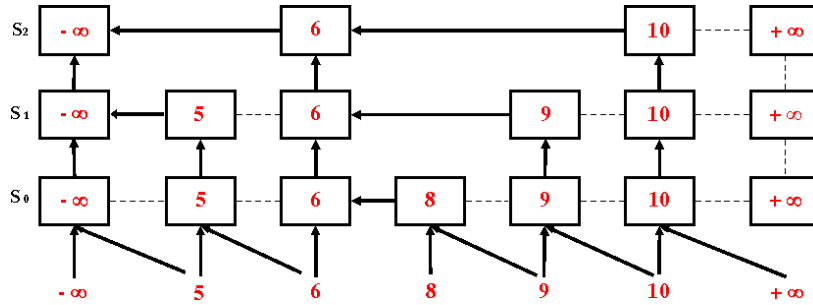


Figure 1.4: Authenticated Skip List: Flow of the computation of the hash values labeling the nodes of the skip list of Fig.1.2. Nodes where hash functions are computed are drawn with thick lines. The arrows denote the flow of information, not links in the data structure.

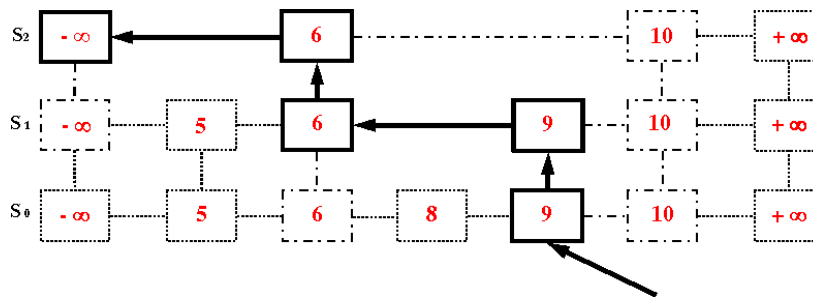


Figure 1.5: Values needed to authenticate the result of a query.

then the user is assured of the validity of the answer at the time given by the timestamp.

Authenticated Data Structures, and in particular Authenticated Skip Lists, could be used to answer dictionary-based (or membership) queries, as well as more advanced queries such as in relational databases or graph queries. For the former, it has been used to design efficient public-key revocation system. For the latter, graph and geometric searching applications use this datastructure as the fundamental component.

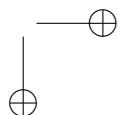
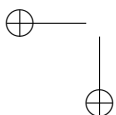
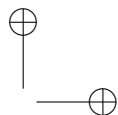
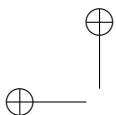
## 1.4 Overview and Thesis Structure

In Chapter 2, we introduce a general method, based on the usage of typical DBMS primitives, for maintaining authenticated relational tables. The authentication process is managed by an application external to the DBMS, that stores just one hash information of the authentication structure. The method exploits techniques to represent hierarchical data structures into relational tables and queries that allow an efficient selection of the elements needed for authentication. In Chapter 3, we present an extension of the techniques introduced in the previous chapter to authenticate the integrity and completeness of query results on relational tables with conditions on different fields at the same time. The method exploits concurrent processing techniques that allow to bind the complexity to the most selective field in the query. Also, this method allows to save storage space because it stores just an authenticated index for each field in the table instead of a different index for each possible combination of the fields without add any restrictive condition on the query. Further, this approach allows to perform authenticated join operation in a more efficient way as we show in an extensive set of test. In Chapter 4, we present a general method and a practical prototype application for verifying the integrity of files in an untrusted network storage service. The verification process is managed by an application running in a trusted environment (typically on the client) that stores just one cryptographic hash value of constant size, corresponding to the digest of an authenticated data structure. The proposed service can sit on top of any storage service since it is transparent to the storage technology used. Experimental results show our integrity verification method is efficient and practical for network storage systems. In Chapter 5, we give a preliminary survey of approaches to the visualization of computer security concepts that use graph drawing techniques. This chapter In Chapter 6, we present a visual representation of access control permissions in a standard hierarchical

#### 1.4. OVERVIEW AND THESIS STRUCTURE

13

file system. Our visualization of file permissions leverages treemaps, a popular graphical representation of hierarchical data. In particular, we present a visualization of access control for the NTFS file system that can help a non-expert user understand and manipulate file system permissions in a simple and effective way. While our examples are based on NTFS, our approach can be used for many other hierarchical file systems as well. Parts of this dissertation have previously appeared as [23, 43, 44, 97] or have been submitted for publication in conferences or journals.



## Chapter 2

# Authenticated Relational Tables and Authenticated Skip Lists

### 2.1 Introduction

We consider the following scenario. A user needs to store data in a relational database, where the Data Base Management System (DBMS) is shared with other users. For example, the DBMS is available on-line through the Web, and anybody in the Internet can store and access data on it. Nowadays, there are many sites providing services of this type [8, 80, 86, 108] and the literature refers to such facilities as to *outsourced databases* [54, 73, 95].

When the database is accessed, the user wants to be sure on the integrity of her/his data, and wants to have the proof that nobody altered them.

Of course, accessing the DBMS is subject to authentication restrictions, and the users must provide credentials to enter. However, the user might not trust the DBMS manager, or the site that provides the service, or even the DBMS software. Extending the argument, the same problem can be formulated even in terms of a traditional database. Also in this case, with the current technologies, although DBMSes put at disposal logs of the performed transactions and other security features, for a user it is somehow impossible to be completely sure that nobody altered the data.

A first attempt for the user to be sure of the authenticity of the data is to put a signature on each  $t$ -uple of each relational table of the database. Unfortunately, this technique does not provide enough security. In fact, adversaries could remove some  $t$ -uples and the user would not have any evidence of this.

Another straightforward possibility would be to sign each table as a whole. However, this does not scale-up, and even mid-size tables would be impossible to authenticate.

We propose a method and a prototype for solving the above mentioned problem. Namely, for each relational table  $R$  of the user we propose to store in an extra relational table  $S(R)$  (in the following *security table*) of the DBMS a special version of authenticated data structure that allows to verify the authenticity of  $R$ .

With this approach, if the user wants to have the proof of authenticity of  $R$ , it is sufficient to check the values of a few elements stored in  $S(R)$ . On the other hand, if the user updates  $R$ , only a few variations on  $S(R)$  are needed to preserve the proof of authenticity. We also propose efficient techniques to manage and to query  $S(R)$  and show the practical feasibility of the approach.

Observe that the proposed approach is completely independent on the specific adopted DBMS and can be implemented into an extra software layer or either a plug-in, under the sole responsibility of the user. The authentication process is managed by an application external to the DBMS that stores just a constant size ( $O(1)$  with respect to the size of  $R$ ) secret. The method does not require trust in the DB manager or DBMS.

## 2.2 The Reference Model

A user stores a relational table  $R$  into a DBMS. The user would like to perform the usual relational operations on  $R$ , namely, would like to select a set of  $t$ -uples, to insert elements, and to delete elements. The user wants to verify that a query result is authentic. The amount of information that the user has to maintain in a secure environment to be certain of the authenticity of the answer should be kept small (ideally constant size) with respect to the size of  $R$ .

We propose to equip  $R$  with an authenticated skip list  $A$  to guarantee its integrity. Of course, there are at least two approaches for implementing  $A$ . Either  $A$  is stored in main memory within an application controlled by the user, or  $A$  is stored into the same DBMS storing  $R$ . We follow the second approach. Namely, we investigate how to efficiently store  $A$  into a further relational table  $S(R)$ , called *security table*, used only for that purpose. Fig. 2.1 shows a relational table, an authenticated skip list for its elements, and the implementation of the skip-list into a second relational table.

There are two options. We call them the *coarse-grained* and the *fine-grained*

## 2.2. THE REFERENCE MODEL

17

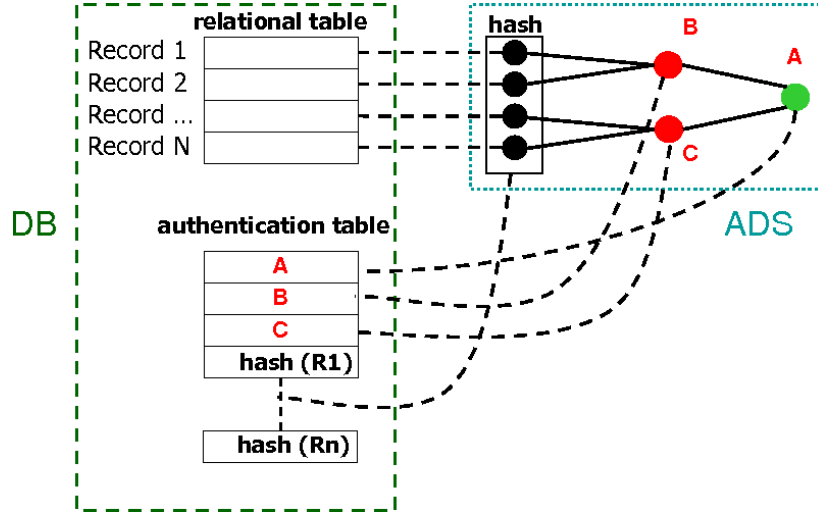


Figure 2.1: A relational table and its security table.

approach.

What we call coarse-grained approach is probably the most natural way to represent an authenticated skip list  $S$  inside a relational table  $S(R)$ . Namely, it consists of storing each element of  $S$  inside a specific record of  $S(R)$ . On the other hand, the fine-grained approach shifts the attention on a smaller element of  $S$ . It consists of storing each level of an element of  $S$  inside a record of  $S(R)$ .

In order to visualize the coarse-grained approach, it is effective to think at  $S$  in terms of a “quarter clockwise rotation”. As an example, Table 2.1 is a coarse-grained representation of the authenticated skip list of Fig. 2.2.

More precisely, the fields of Table 2.1 have the following meaning.

- **Key**: The value of an element of  $S$ . It can be any type of value, not only a number, but on such a type a total ordered must be defined.
- **Prv  $n$  - Nxt  $n$** : Pointers to the previous and to the next element in  $S$ , for each level  $n$ .
- **Hash  $n$** : Information needed to authenticate  $S$ , stored at each level  $n$ .

18 CHAPTER 2. AUTHENTICATED RELATIONAL TABLES AND AUTHENTICATED SKIP LISTS

Each element of  $S$  has a height, that is, the number of nodes with the same value of key that constitute an element of  $S$ , that is randomly determined. This is the main trade-off of this technique, because on one hand this kind of representation has the property to maintain the identity between the number of records in  $S(R)$  and the elements present in  $S$ , but on the other hand it has an overhead in the size of the table, because each record has a number of fields equal to the highest  $S$  in  $A$ . This is necessary because we do not know the height of a new  $S$  and then we have to arrange  $S(R)$  for worst cases, when an  $S$  is at the highest level. So, we must pad with “null” values the fields that do not reach the highest level.

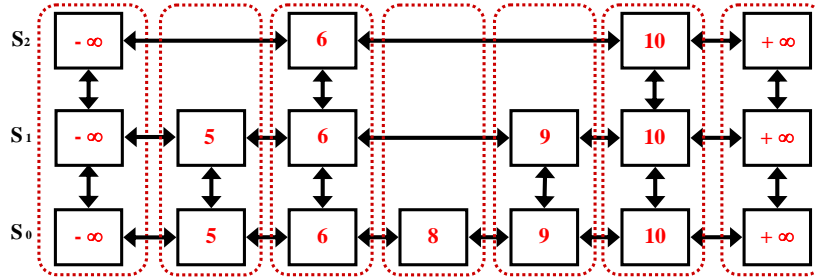


Figure 2.2: Storing a Skip List inside a Relational Table.

Once stated how to represent  $S$  inside the security table  $S(R)$ , we developed methods to perform in  $S$  a set of authenticated relational operations, without the need to load in main memory the whole  $S(R)$ . Performing authenticated operations on  $R$  requires the usage of queries that retrieve all the elements that are needed to compute the authentication path. Such elements are spread on all  $S(R)$ . The main requirements in devising such queries are:

- The need to build queries that retrieve only the authentication elements that are strictly necessary, to reduce, as much as possible, the amount of required memory.
- The need of fast queries that allow to authenticate a result with a small time overhead. In this respect it is meaningful to minimize the number of used queries.

It is important to perform such queries using only standard SQL. In fact, our model does not allow any modification of the DBMS engine. Also, think-

## 2.2. THE REFERENCE MODEL

19

Key	Hash 0	Prv 0	Nxt 0	Hash 1	Prv 1	Nxt 1	Hash 2	Prv 2	Nxt 2
$-\infty$	$f(-\infty, 5)$	<i>null</i>	5	$f(-\infty, f(5))$	<i>null</i>	5	$f(f(-\infty), f(6))$	<i>null</i>	<b>6</b>
5	$f(5, 6)$	$-\infty$	6	$f(f(5), f(6))$	$-\infty$	6	<i>null</i>	<i>null</i>	<i>null</i>
<b>6</b>	$f(6, f(8))$	5	8	$f(f(6), f(9))$	5	<b>9</b>	$f(f(6), f(10))$	10	$-\infty$
8	$f(9, 6)$	9	6	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>	<i>null</i>
<b>9</b>	$f(9, 10)$	8	10	$f(9, 10)$	<b>6</b>	<b>10</b>	<i>null</i>	<i>null</i>	<i>null</i>
<b>10</b>	$f(10, f(+\infty))$	9	$+\infty$	$f(f(10), f(+\infty))$	9	$+\infty$	$f(f(10), f(+\infty))$	6	$+\infty$

Table 2.1: A coarse-grain representation of an authenticated skip list into a relational table. In bold face the elements necessary to authenticate element 9.

CHAPTER 2. AUTHENTICATED RELATIONAL TABLES AND  
AUTHENTICATED SKIP LISTS

ing in terms of SQL allows the identification of a precise interface between an authentication tool based on our techniques and the DBMS, allowing its implementation in terms of a plug-in. The main idea here is to use an algorithm that retrieves the authentication elements, starting from the knowledge of the value  $K$  to authenticate:

1. We perform a query that loads in memory all the records that are not *null* at top level and that have a value smaller than  $K$ .
2. We select the greatest element in the query result (that is the predecessor of  $K$  at the top level).
3. We perform an interval query on the elements (that are not *null*) at the immediately lower level, with the following range: from the element retrieved in the previous step to the element stored in its field next to the top level.
4. We repeat the steps 2 – 3 until we reach level 0.

In order to understand which elements are loaded in main memory by queries of the algorithm, it is effective to think at a shape like a “funnel” that has its stem on  $K$ . See Fig. 2.3. The loaded elements are those that “touch” the funnel.

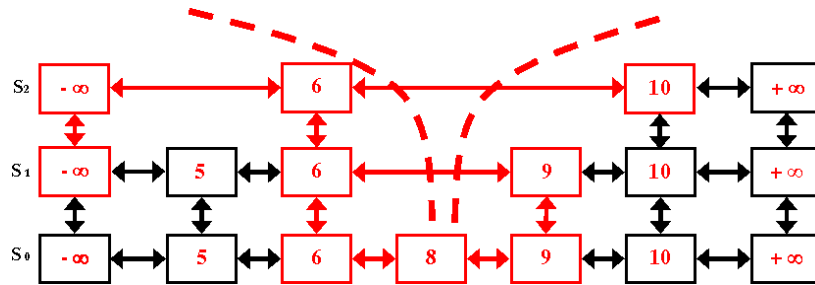


Figure 2.3: Loaded elements in an authentication query.

Note that the number of queries that is needed to retrieve the authentication root path is proportional to the number of levels in  $S$ , that is logarithmic in the number of elements that are currently present in  $S$ .

### 2.3 A Fine Grained Approach

This approach stores inside each record a node instead of an element of  $S$ . A node is an invariant-size component in  $S$ . Hence, it can be stored in a record with a fixed number of fields, independently on the number of elements stored in  $S$ . More precisely, in this case the fields of  $S(R)$  have the following meaning:

- **Key**: value of an element of  $S$ ;
- **Level**: height of an element of  $S$ , that is the number of the lists that the element belongs to;
- **prvKey-nxtKey**: pointers to the previous and to the next element of  $S$  at the same level;
- **parentLvl-parentKey**: pointer to the parent element in the path of authentication; it is needed to allow the retrieval of the root path;
- **Hash**: information needed for the authentication, performed with the method used in  $S$  [34].

The direct storage of  $S$  nodes significantly reduces the space overhead, that it is typical of the coarse grain approach. In fact, in this case there is no need to store *null* values.

This approach allows the usage of very efficient techniques to manage  $S(R)$  dynamically and securely. The method we adopt is based on the *nested set* method for storing hierarchical data structures inside adjacency lists, that in turn fit well into relational tables [16] See Fig. 2.4 and Tab. 2.2.

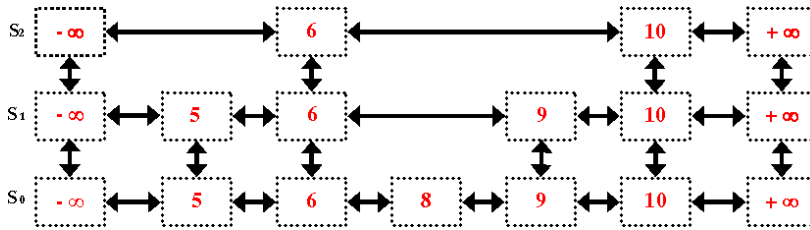


Figure 2.4: Storing a Skip List inside a Relational Table. A Fine Grained Approach

22 CHAPTER 2. AUTHENTICATED RELATIONAL TABLES AND  
AUTHENTICATED SKIP LISTS

Key	Level	prvKey	nxtKey	parentLvl	parentKey	Hash
$-\infty$	<b>2</b>	<i>null</i>	6	<i>null</i>	<i>null</i>	$f(f(-\infty), f(\mathbf{6}))$
$-\infty$	<b>1</b>	<i>null</i>	5	2	$-\infty$	$f(f(-\infty), f(\mathbf{5}))$
$-\infty$	<b>0</b>	<i>null</i>	5	1	$-\infty$	$f(f(-\infty), \mathbf{5})$
5	1	$-\infty$	6	1	$-\infty$	$f(5, 6)$
5	0	$-\infty$	6	1	5	$f(5, 6)$
<b>6</b>	<b>2</b>	$-\infty$	<b>10</b>	<b>2</b>	$-\infty$	$f(f(\mathbf{6}), f(\mathbf{10}))$
<b>6</b>	<b>1</b>	5	<b>9</b>	<b>2</b>	<b>6</b>	$f(f(\mathbf{6}), f(\mathbf{9}))$
<b>6</b>	<b>0</b>	5	8	1	6	$f(\mathbf{6}, f(\mathbf{8}))$
8	0	6	9	0	6	$f(8, 10)$
<b>9</b>	<b>1</b>	<b>6</b>	<b>10</b>	<b>1</b>	<b>6</b>	$f(\mathbf{9}, \mathbf{10})$
<b>9</b>	<b>0</b>	8	10	1	9	$f(\mathbf{9}, \mathbf{10})$
<b>10</b>	<b>2</b>	6	$+\infty$	2	6	$f(\mathbf{10}, f(+\infty))$
<b>10</b>	<b>1</b>	9	$+\infty$	2	10	$f(\mathbf{10}, f(+\infty))$
10	0	9	$+\infty$	2	10	$f(10, f(+\infty))$

Table 2.2: A fine grain representation of an authenticated skip list into a relational table. In bold the elements necessary to authenticate element 9

## 2.4 Exploiting Nested Sets

The problem of storing hierarchical data structures inside relational tables has been already studied in database theory [19, 63]. The solution that we exploit is due to Celko [16], that shows a method to store a tree inside a relational table. Such a method is based on augmenting the table with two extra fields.

In order to understand what is a nested set, it is effective to think at the nodes of the tree as circles and to imagine that the circles of the children are nested inside their parent. The root of the tree is the largest circle and contains all the other nodes. The leaf nodes are the innermost circles, with nothing else inside them. The nesting shows the hierarchical relationship.

The two extra fields have the role of left and right boundaries of the circle and allow to represent the nesting of the hierarchy.

Unfortunately, skip lists are not *trees* but a directed acyclic graph. Hence, we have to extend the nested set method to this different setting. Table 2.3 illustrates how the fine-grained approach can be equipped with nested-sets features. Observe the Left and Right fields that represent the boundaries of the “circles”. Fig. 2.5 shows the correspondence between boundaries and nodes of the skip-list. The figure shows also a root path.

Now we show one of the features of the proposed approach. Namely, we argue that, in order to authenticate an element of a relational table  $R$ , we need

#### 2.4. EXPLOITING NESTED SETS

23

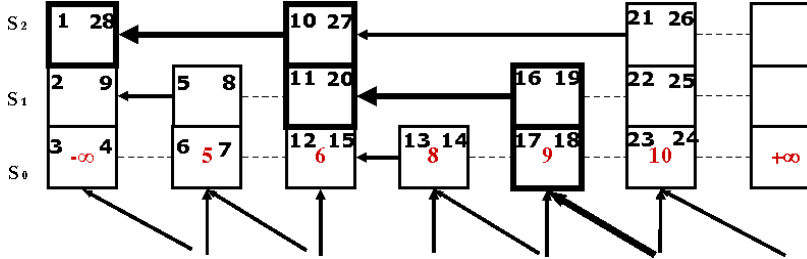


Figure 2.5: An ADS and its Nested Set. Thick lines show the authentication root path for element 9.

just one query on  $S(R)$ . Such a query is used to retrieve the complete *root-path* and all its *sibling* elements. Observe that, authenticating an element in an ADS requires a number of steps that is logarithmic (worst case or average case) in the number of the elements while this logarithmic dependence does not yield a logarithmic number of queries in our case but a constant number of queries. We make the argument using an example. The following query uses directly

Key	Level	prvKey	nxtKey	parentLvl	parentKey	Left	Right
$-\infty$	<b>2</b>	<i>null</i>	<b>6</b>	<i>null</i>	<i>null</i>	<b>1</b>	<b>28</b>
$-\infty$	<b>1</b>	<i>null</i>	<b>5</b>	<b>2</b>	$-\infty$	<b>2</b>	<b>9</b>
$-\infty$	<b>0</b>	<i>null</i>	<b>5</b>	<b>1</b>	$-\infty$	<b>3</b>	<b>4</b>
<b>5</b>	<b>1</b>	$-\infty$	<b>6</b>	<b>1</b>	$-\infty$	<b>5</b>	<b>8</b>
<b>5</b>	<b>0</b>	$-\infty$	<b>6</b>	<b>1</b>	<b>5</b>	<b>6</b>	<b>7</b>
<b>6</b>	<b>2</b>	$-\infty$	<b>10</b>	<b>2</b>	$-\infty$	<b>10</b>	<b>27</b>
<b>6</b>	<b>1</b>	<b>5</b>	<b>9</b>	<b>2</b>	<b>6</b>	<b>11</b>	<b>20</b>
<b>6</b>	<b>0</b>	<b>5</b>	<b>8</b>	<b>1</b>	<b>6</b>	<b>12</b>	<b>15</b>
<b>8</b>	<b>0</b>	<b>6</b>	<b>9</b>	<b>0</b>	<b>6</b>	<b>13</b>	<b>14</b>
<b>9</b>	<b>1</b>	<b>6</b>	<b>10</b>	<b>1</b>	<b>6</b>	<b>16</b>	<b>19</b>
<b>9</b>	<b>0</b>	<b>8</b>	<b>10</b>	<b>1</b>	<b>9</b>	<b>17</b>	<b>18</b>
<b>10</b>	<b>2</b>	<b>6</b>	$+\infty$	<b>2</b>	<b>6</b>	<b>21</b>	<b>26</b>
<b>10</b>	<b>1</b>	<b>9</b>	$+\infty$	<b>2</b>	<b>10</b>	<b>22</b>	<b>25</b>
<b>10</b>	<b>0</b>	<b>9</b>	$+\infty$	<b>2</b>	<b>10</b>	<b>23</b>	<b>24</b>

Table 2.3: A representation of an authenticated skip list into a relational table using nested set. In bold the key value and the *left* and *right* fields. The 2 extra fields added are needed for fast queries.

the value of the element to authenticate. The example is for the authentication

## CHAPTER 2. AUTHENTICATED RELATIONAL TABLES AND AUTHENTICATED SKIP LISTS

24

of element 9.

```
SELECT      *
FROM        skiplist
WHERE Left <= (SELECT      Left
                FROM        skiplist
                WHERE        key = 9 AND level = 0)
AND Right >= (
                SELECT      Right
                FROM        skiplist
                WHERE        key = 9 AND level = 0);
```

The above query retrieves only the authentication root-path starting from 9. To validate 9 we have to retrieve also all sibling nodes of the root-path. This is possible by using two subqueries that retrieve all elements that are:

- in the fields nxtKey of the root-path;
- on the level below and with the same key of the root-path.

Using this method we built a quick algorithm to get the complete authentication path needed to validate a table interrogation, using only one query, that is that all concurrency problems related to selection queries will be managed by the DBMS. Also, it is possible to modify the query in order to retrieve all the information needed to authenticate all the  $t$ -uples obtained by a Select with just one query.

### 2.5 Experimental Evaluation

This section shows the experimental results obtained using a prototype implementation of the techniques presented in the previous sections. The Hardware architecture where tests have been performed consists of quite common laptop with following features:

- cpu intel©centrino<sup>TM</sup> duo T2300 (1.66 GHz, 667 FSB);
- RAM 1.5 Gb DDR2
- HDD 5,400 rpm Serial ATA

The Software architecture consists of following elements:

## 2.5. EXPERIMENTAL EVALUATION

25

- Microsoft©Windows<sup>TM</sup>XP Tablet edition 2005;
- Java<sup>TM</sup>version 1.5
- MySql JDBC Connector Java-bean 5.03
- MySql DBMS version 4.1

The data sets for tests have been chosen with a scale from 10,000 to 1,000,000 of elements. Such elements were sampled at random from a set 10 times larger. All values presented in this section have been computed as average of the results of 5 different tests. The elements in each test are a sample, randomly selected, composed of  $\frac{1}{1000}$  of the entire set. All times are in *milliseconds*. All tests show the *clock-wall* time.

The first test is about the authentication of a single value inside a relational table. Table 2.4 shows the results of the authentication of a single element inside different size authenticated tables, stressing the differences between coarse grain and fine grain approaches. Tests are about the following measures:

- **RAM**: the time to validate a value in main memory;
- **DB**  $\rightarrow$  **RAM**: the time to load in main memory from a secondary memory storage system (e.g., a hard disk), the elements necessary to validation;
- **NODES**: the numbers of elements loaded from the database in main memory;
- **STEPS**: the numbers of elements actually used in the authentication process, the difference between NODES value and this value shows the overhead of the elements loaded in main memory.

The results showed above are very similar to those obtained from the authentication of an element not-present in the table. In fact it is sufficient to check the previous and the next element of the value that is not present to proof the element lack.

The second test is about the insertion of a single value inside an authenticated relational table. The table 2.5 shows the results of the insertion of a single element inside different size authenticated tables using only coarse grain approach. Tests concern the following measures:

- **RAM**: the time to insert in main memory a value;

CHAPTER 2. AUTHENTICATED RELATIONAL TABLES AND  
AUTHENTICATED SKIP LISTS

CHECK	10,000		100,000		1,000,000	
	Coarse	Fine	Coarse	Fine	Coarse	Fine
RAM	0	0	0	0	0	0
DB $\rightarrow$ RAM	36	11	252	42	2680	377
NODES	35	27	44	31	57	43
STEPS	25	27	33	30	39	41

Table 2.4: Test results for validation of an element inside different size tables. All the results are in *ms*. Times for fine- and coarse-grained approaches.

- **DB  $\rightarrow$  RAM**: the time to load in main memory from a secondary memory storage system (e.g., a hard disk), the elements necessary to insertion;
- **RAM  $\rightarrow$  DB**: the time to store in secondary memory the elements updated in main memory;

INSERT	10,000	100,000	1,000,000
RAM	0	0	0
DB $\rightarrow$ RAM	32	260	2605
RAM $\rightarrow$ DB	14	26	26
Tot. Time	46	286	2631

Table 2.5: Test results for insertion of an element inside a different size tables. Using coarse grain approach. All results are in *ms*.

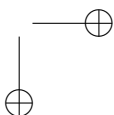
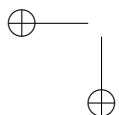
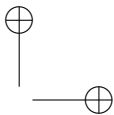
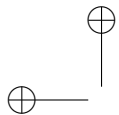
Methods that allow to delete and modify an element inside an authenticated table are similar to times showed for insertion operation.

The obtained experimental results put in evidence the feasibility of the approach. In fact, the time for answering a query is comparable to the one obtained in a non authenticated setting. The fine-grained approach, based on Celko techniques, shows much better performance with respect to the coarse-grained one.

## 2.6 Conclusions

We have described methods that allow a user to verify the authenticity and completeness of simple queries results, even if the database system is not trusted. The overhead for the user is limited at storing only a single hash value. Our work is the first to design and evaluate techniques for authenticated skip list that are appropriate to a relational database, and the first to prove the feasibility of authenticated skip list for integrity of databases.

The security of the presented method is based on the reliability of *ADSeS*. There are many works [12, 34, 51, 65] in the literature that demonstrate that the security of *ADS* is based on the difficulty to find useful collisions in a cryptographic hash function. So all the security relies on the effectiveness of hash functions. The prototype used for the experiments uses commutative hashing. In [34] it is demonstrated that commutative hashing does not augment the possibility to find a collision in the used hash function.



## Chapter 3

# Multi-Column Relational Query Results Authentication

### 3.1 Introduction

Advances in networking technologies and continued spread of the Internet jointly with cost-effective offers have triggered a trend towards outsourcing data management and information technology needs to external service providers. Database outsourcing is a known evidence of this trend. The outsourced database (ODB) users rely on the infrastructure of the provider, which include hardware, software and manpower, for the storage, maintenance, and retrieval of their data. That is, a company stores all its data, included confidential information, at an external service provider, that is generally not fully trusted. The final goal for the user is to use the ODB service as an in-house database, without taking care of the untrusted server at the provider's side. Actually, this approach involves several security issues that range from confidentiality preservation to integrity verification. This chapter studies protocols for authenticating the integrity of ODB in ways that achieve high security and efficiency level. Our approach exploits the technique described in [23] that allows the user to have the proof of authenticity of a query result by checking only a few of elements stored in an authenticated answer with a complexity  $O(\log n)$  with  $n$  the number of records in the original table. At the best of our knowledge the existing techniques [23, 12, 68, 82] allow to authenticate one-dimensional range search queries, that is, ask the database to report those records having values of a certain field within given bounds. Instead our goal is to design se-

curity protocol that allows to efficiently authenticate multi-dimensional range search queries, that is to say, ask the database to report those records having values on more than one field within given values.

Cod.	Student	Homework	Grade
203	White Anne	02/01/08	A
574	Brown Jake	02/01/08	A
461	Brown Luke	02/04/08	B
530	White Mark	NULL	C
405	Black Lucy	02/01/08	E
501	Smith Joe	NULL	NULL
224	Ferley Peter	03/05/08	D
525	Cornwell Sharon	NULL	NULL
416	Baxter Frank	02/04/08	A
489	Moore Mark	NULL	NULL

Table 3.1: *Scores*, the original table with exam results.

An existing approach to solve this problem in one-dimension is to store, for each relational table  $R$  of the user, an extra relational table  $S(R)$  (in the following security table) is a special type of authenticated data structure that allows to verify the authenticity of  $R$ , see as an example table 3.1. A straightforward extension to the multi-dimensional case would be to authenticate all the combinations  $O(2^n)$  of the  $n$  table fields, namely for the table 3.1 it is the power set of the table fields.

$$\begin{aligned} \mathcal{P}(\text{Cod.}, \text{Student}, \text{Homework}, \text{Grade}) = \\ \{ \{ \text{Cod.} \}, \{ \text{Student} \}, \{ \text{Homework} \}, \{ \text{Grade} \}, \\ \{ \text{Cod.}, \text{Student} \}, \{ \text{Cod.}, \text{Homework} \}, \{ \text{Cod.}, \text{Grade} \}, \\ \{ \text{Student}, \text{Homework} \}, \{ \text{Student}, \text{Grade} \}, \{ \text{Homework}, \text{Grade} \}, \\ \{ \text{Cod.}, \text{Student}, \text{Homework} \}, \{ \text{Cod.}, \text{Student}, \text{Grade} \}, \\ \{ \text{Cod.}, \text{Homework}, \text{Grade} \}, \{ \text{Student}, \text{Homework}, \text{Grade} \}, \\ \{ \text{Cod.}, \text{Student}, \text{Homework}, \text{Grade} \} \} \end{aligned}$$

Unfortunately, this does not scale-up, and even tables with a few attributes would be impossible to authenticate. We propose a method and a prototype for solving the above mentioned problem. Namely, for each dimension (field) of a relational table  $R$  of the user we propose to store only one security table. With the proposed approach, if the user wants the proof of authenticity of a query with conditions on different fields of  $R$  it is sufficient to check the values of a

### 3.2. THE AUTHENTICATION PROBLEM

31

few elements stored in  $S(R)$  of just the most selective attribute of the query. On the other hand, if the user updates  $R$ , the variations needed to preserve the proof of authenticity on the  $S(R)$  of each fields can be performed with a strong parallelism. So we obtain a negligible time overhead in comparison with the one dimension approach. Further, the authentication mechanism introduced into this chapter allows to efficiently authenticate join operations. Observe that the proposed approach is completely independent on the specific adopted DBMS and can be implemented into an extra software layer or either a plug-in, under the sole responsibility of the user. The authentication process is managed by an application external to the DBMS that stores just a secret string of constant size ( $O(1)$  with respect to the size of  $R$ ). The method does not require trust in the DB manager or DBMS.

### 3.2 The Authentication Problem

A professor stores student scores of his course inside a digital table (e.g. relational table, flow chart) hosted in the department computer server. The professor would like to use this information to record the exam grade for each student. So, he queries the table and trusts obtained scores as if they came from the original table. Actually, if you trust the scores as a consequence you have to trust the entire *chain* that manages his data, in example: software used, department network, university technical staff, etc. So each ring of the chain could be a potential weakness point and we can consider following threats for exam scores.

- **integrity:** a rogue student could modify his exam grade in the table and the professor does not have any tool to realize that the grade is no longer the same that was on the original table.
- **completeness:** the professor prints a *blacklist* with the students that did not submit the homework properly. So, they can not take the exam. What happens if he get a list incomplete, probably some rogue students can take the exam. Unfortunately in this case too he does not have any tool to check that the table has the same number of elements of the original one.
- **multi dimensional query:** the professor wants to check if there are some mistakes. So he queries the table to verify if there are students that did not submit the homework properly and that have passed the exam. In this case too the professor has to check integrity and completeness

with conditions that have to be verified on different fields at the same time.

- **join query:** the professor normalizes *Scores* table and obtains two different tables *JScores* and *Students*. He would like to make a phone call to the students that have received a score less than *C*. To retrieve this information he performs a query with a *join* on both tables and the condition on the *grade* field. Also in this case the professor has to check integrity and completeness of the resulting table because for instance he can receive a partial number of records from the *join*.

The goal of our work is to devise a solution for checking the correctness of query answers on multi-dimensional datasets.

To be more precise, we can use as reference the exam scores table and we can imagine that the professor performs some queries on it. The professor would have a validation function that allows to check correctness of the query result, namely to check if the result is the same that he would obtain performing the same query on the original table. Original table means a not tampered version of the table. In the following we show some examples performed on Table 3.1 of how our model works.

- **Query 1:** The professor obtains the list of exam scores to publish.

```
SELECT    *
FROM      scores
WHERE     Grade = 'A' OR Grade = 'B' OR Grade = 'C';
```

In this scenario we have a different result between query performed on the original table and the same one performed on a tampered table. It is easy to notice this change by comparing query results, because you know the result from the original table. In our approach we validate directly query result: the professor, if the validation process result is affirmative, is sure that what he publishes are the correct exam scores. So the professor can notice that student *White Mark* tampered the exam scores table by changing his grade from **C** to **A**. Further, if another student changes his grade from **E** to **D** the professor does not get any error and that is right because with query 1 he does not care about the insufficient grades.

- **Query 2:** The professor obtains the list of students that did not submit the homework properly.

### 3.2. THE AUTHENTICATION PROBLEM

33

[Correct scores list.]	Cod.	Student	Homework	Grade
	203	White Anne	02/01/08	A
	574	Brown Jake	02/04/08	A
	461	Brown Luke	02/04/08	B
	530	White Mark	NULL	<b>C</b>
	416	Baxter Frank	02/04/08	A

[Rogue scores list.]	Cod.	Student	Homework	Grade
	203	White Anne	02/01/08	A
	574	Brown Jake	02/04/08	A
	461	Brown Luke	02/04/08	B
	530	White Mark	NULL	<b>A</b>
	416	Baxter Frank	02/04/08	A

Table 3.2: integrity problem

```

SELECT    Cod., Student, Homework
FROM      scores
WHERE     Homework IS NULL ;

```

[Correct Blacklist.]	Cod.	Students	Homework
	530	White Mark	NULL
	501	Smith Joe	NULL
	525	Cornwell Sharon	NULL
	489	Moore Mark	NULL

[Rogue Blacklist.]	Cod.	Students	Homework
	501	Smith Joe	NULL
	525	Cornwell Sharon	NULL
	489	Moore Mark	NULL

Table 3.3: completeness problem

In this scenario too we have different query results between the original

table and tampered one. It is easy to verify this change by comparing query results, because one record is missing. This query concerns the completeness checking problem of the result. In our approach we check directly query result: if the professor does not receive any error is sure that no record misses in the student blacklist that can not take the exam. So the professor can notice that student *White Mark* removed his record from the blacklist table. Moreover, if there is any change in the grade field, the professor does not get any error and that is right, because with query 2 he is not interested in this field.

- **Query 3:** The professor would like to check if for instance there are some problems in the last exam scores before to record final scores.

```
SELECT      *
FROM    scores
WHERE    Homework is NULL and Grade IS NOT NULL;
```

[Correct Multidimensional Query]			
Cod.	Student	Homework	Grade
530	White Mark	NULL	C
[Rogue Multidimensional Query]			
Cod.	Student	Homework	Grade

Table 3.4: multidimensional problem

This example also shows a different query result between the query performed on the original table and the same query performed on a tampered table. In this case we have conditions on more than one field to verify at the same time. This is the multi dimension query problem. The professor, if does not receive any error, is sure that he can find all existent inconsistencies (if any). So the professor can notice that student *White Mark* cheated at the exam. Besides, this method allows to verify also the soundness of an empty result.

- **Query 4:** The professor would like to make a phone call to the students that have received a score less than *C*.

### 3.2. THE AUTHENTICATION PROBLEM

35

Cod.	Homework	Grade
203	02/01/08	A
405	02/01/08	E
224	03/05/08	D

Table 3.5: *Jscores*, the table with exam results.

Cod.	Student	Phone
203	White Anne	555-123-1234
405	Black Lucy	555-456-2348
224	Ferley Peter	555-768-3457

Table 3.6: *Students*, the table with student phone number.

```
SELECT *
FROM Jscores NATURAL JOIN Students
WHERE Grade > 'C'
```

[Correct Natural Join Table.]

Cod.	Student	Phone	Homework	Grade
405	Black Lucy	555-456-2348	02/01/08	E
224	Ferley Peter	555-768-3457	03/05/08	D

[Rogue Natural Join Table.]

Cod.	Student	Phone	Homework	Grade
405	Black Lucy	555-456-2348	02/01/08	E

Table 3.7: *Join Table*, Natural Join Table result between Table 3.6 and Table 3.5.

In this scenario we have different query results because the Natural Join result between Table 3.6 and Table 3.5 does not work correctly. It is easy to verify this change by comparing the results in Table 3.7 because one record is missing. This query concerns the completeness checking

problem of the *Join* result. In our approach we check directly query result: if the professor does not receive any error is sure that no record misses in the student blacklist that are not allowed take the exam. So the professor can notice that student *Ferley Peter* removed his record from the join result table.

The model that we describe in this chapter allows to authenticate, beyond *selection*, *insertion* and *deletion* queries and moreover we developed an algorithm to verify set theory operations on a relational table, namely: *union*, *intersection* and *difference*. It is also available a method to check authenticity of *max* and *min* of a query result. The well known *join* operation authentication takes advantage of this method.

### 3.3 Architecture

During the technical architecture development we followed the model introduced in [23]. Namely, a user stores data in a relational table  $R$  and he would verify the authenticity of a query result performed on  $R$ . The main idea in [23] is: considering  $R$  as a table whose records are sorted by its attribute  $A$ , is possible to create an authenticated skiplist structure, which can be stored in the same or in a different *DBMS*, whose purpose is to authenticate records in  $R$  under the condition they are sorted by  $A$ . This approach allows to authenticate only mono dimensional queries performed on  $A$ . Our approach extend the previous one to multidimensional queries. Namely, we authenticate the result of queries with conditions that can be true on every attribute or possible combination of attributes on  $R$ .

Key	rowhash	Hash 0	k Prv0	rh Prv0	k Nxt0	rh Nxt0
224	$h(r_7)$	$h(h(224, h(r_4)), h(416, h(r_9)))$	203	$h(r_1)$	416	$h(r_9)$
416	$h(r_9)$	$h(h(416, h(r_9)), f(461, h(r_3)))$	224	$h(r_7)$	461	$h(r_3)$
461	$h(r_3)$	$h(h(461, h(r_3)), h(489, h(r_7)))$	416	$h(r_9)$	489	$h(r_10)$

Table 3.8: Skiplist table with rowhash

To solve the problem in a multidimensional environment we add one vertical dimension to ensure completeness and one horizontal dimension to ensure integrity.

### 3.3. ARCHITECTURE

37

- **vertical dimension:** we introduce it to address the completeness (*query 2*) check problem for each column of  $R$ . A user executes a selection query on different fields of  $R$ . The query result can be checked by using a different authenticated skiplist for each table column. This method stores each skiplist in a different relational table.
- **horizontal dimension:** we introduce it to address the integrity (*query 1*) check problem for each row of  $R$ . We can verify the row integrity using a hash value that we call *rowhash*. This value is calculated using the projection technique introduced in [82].

We use two dimensions to authenticate the entire table because they allow to verify the correctness with granularity of a single element. We introduce a multidimensional extension of the authenticated skip list, that consists in storing the *rowhash* value inside the authenticated skiplist. The new authentication algorithm stores, for each element  $v$  inside a record  $r$ , a label computed accumulating the elements of the set using a commutatively cryptographic hash function  $h$ . That is,  $h(a, b) = h(b, a)$ , this function obtains the same security level of standard hash function used (e.g.: md5, sha 1, sha 256, etc. ) see [34] for details.  $h(r)$  is the *rowhash* of the record  $r$ . For each element  $v$  inside a record  $r_v$  we define label  $f(v, h(r_v))$  in terms of the respective values at nodes  $w = \text{right}(v)$  and  $u = \text{down}(v)$ . If  $\text{right}(v) = \text{null}$ , then we define  $f(v, h(r_v)) = 0$ . The definition of  $f(v, h(r_v))$  in the general case depends on whether  $u$  exists or not for this node  $v$ .

- $u = \text{null}$ , i.e.,  $v$  is on the base level:
  - If  $w$  is a tower node, then
 
$$f(v, h(r_v)) = h(h(\text{elem}(v), h(r_v)), h(\text{elem}(w), h(r_w)))$$
  - If  $w$  is a plateau node, then
 
$$f(v, h(r_v)) = h(h(\text{elem}(v), h(r_v)), f(w, h(r_w))).$$
- $u \neq \text{null}$ , i.e.,  $v$  is not on the base level:
  - If  $w$  is a tower node, then  $f(v, h(r_v)) = f(u, h(r_u))$
  - If  $w$  is a plateau node, then
 
$$f(v, h(r_v)) = h(f(u, h(r_u)), f(w, h(r_w))).$$

Multidimensional extension of authenticated skiplist allows to retrieve the security information concerning *rowhash* directly during the verification of the vertical dimension. That is to say that each element of the table knows the *rowhash* of the record it belongs to. See Fig. 3.1

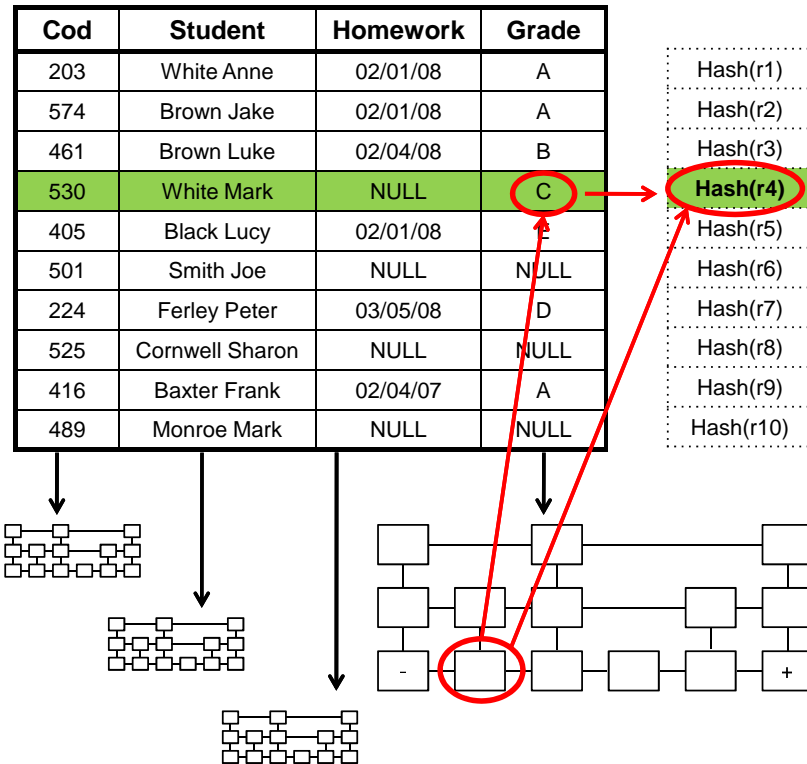


Figure 3.1: A relational table and its security table.

### 3.4 Our Approach

Our approach exploits the developed architecture described in the previous section, that is composed by a multidimensional authenticated skiplist stored in one table like table 3.8 for each column of the original table to authenticate.

### 3.4. OUR APPROACH

39

The user obviously can choose which column to authenticate and this does not change our model. As a consequence he will have to generate only authenticated structure for selected columns.

The proposed algorithm to execute a multidimensional query  $Q$  (on more than one field at the same time) is composed by four steps, see fig. 3.2.

- STEP 1: factorizing  $Q$  to retrieve different queries to apply at each field individually
- STEP 2: performing concurrently all queries factorized in STEP 1, using the first result obtained, the fastest one, and stopping all the other concurrent queries
- STEP 3: creating a view of the entire table filtered using the condition retrieved in step 2
- STEP 4: calculating the *rowhash* for each record presents inside the view in STEP 3 and then checking the completeness of the range query performed on the field selected in STEP 2

### SELECT operation

The first approach for the *select* operation consists in decomposing a complex multidimensional query in several simpler mono-dimensional queries. The main table is then queried in parallel. That is, each column returns its query results, actually a set of records. Each result will be validated with its authentication skiplist column. The intersection among all column results will be the authenticated select query result. This approach is not only straightforward, but it is dependent on the slowest query. Further you must interrogate all authentication skiplists involved in the conditions of the selection query.

To introduce our algorithm we use an example and we suppose to execute Query 3.

In Step 1 we analyze the query and then we decompose in many queries as the conditions expressed on each column. We will obtain then two queries:

- Query 3.1:

```
SELECT *
FROM scores
WHERE Homework IS NULL;
```

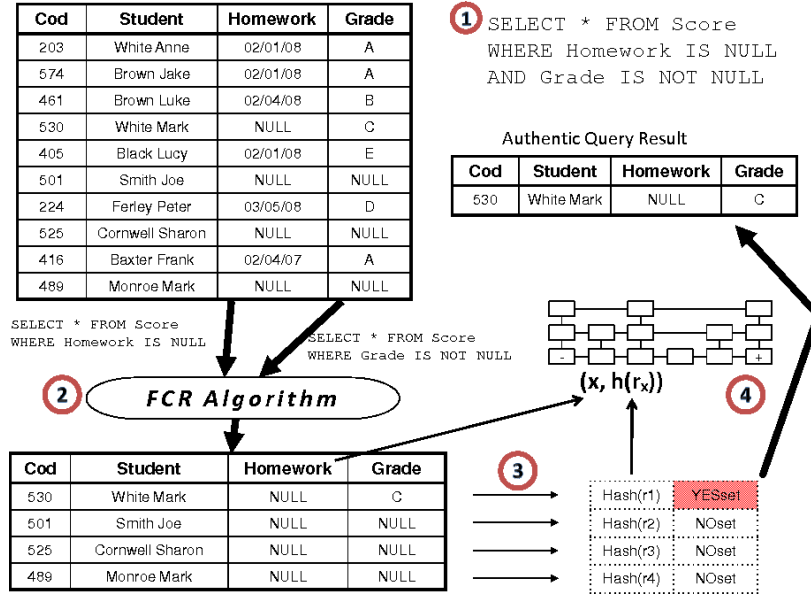


Figure 3.2: Multi Dimensional Authentication Algorithm: the four steps algorithm. *Step 1* parsing the original query to retrieve queries on individual fields. *Step 2* performing concurrently all queries obtained in STEP 1, using the first result obtained, the fastest one, and stopping all the other concurrent queries. *Step 3* showing the result of the previous Step in a view. *Step 4* calculating the rowhash for each record presents inside the view in STEP 3 and then checking the completeness of the range query performed on the field selected in STEP 2.

• Query 3.2:

```
SELECT *
FROM scores
WHERE Grade IS NOT NULL;
```

The multidimensional problem is then decomposed in two simpler mono-dimensional queries. The Step 2 allows the algorithm to individuate the column where the fastest condition is defined. So the query that has answered in less

### 3.4. OUR APPROACH

41

time is detected. The individuated column takes the name of First Column Returned *FCR*. Our approach assumes that probably the view with the first query result returns less records than the others.

Table 3.9 shows that Query 3.1 contains only 4 records, whereas Table 3.10 shows that Query 3.2 has 7 records.

Cod.	Student	Homework	Grade
530	White Mark	NULL	C
501	Smith Joe	NULL	NULL
525	Cornwell Sharon	NULL	NULL
489	Moore Mark	NULL	NULL

Table 3.9: *Step 2*: Resulting Table of query 3.1

Cod.	Student	Homework	Grade
203	White Anne	02/01/07	A
574	Brown Jake	02/01/07	A
461	Brown Luke	02/04/07	B
530	White Mark	NULL	C
405	Black Lucy	02/01/07	E
224	Ferley Peter	03/05/07	D
416	Baxter Frank	02/04/07	A

Table 3.10: *Step 2*: Resulting Table of query 3.2

Step 3 uses the *FCR* that was found in the previous step. The executed query on *FCR* contains an unauthenticated set of records, that is a superset of the result of the original query *Q*. Only a subset of them satisfies all the conditions simultaneously. So we must individuate which records will be returned as result of *Q*. Then we start from a view of the main table produced by the query 3.1. At this point you can filter the obtained view (that is already in main memory) with the other conditions indicated in *Q* on the other fields of the table. So two sets are built: the *YesSet* and *NoSet*. The first group contains the *YesSet* where there are all the records that satisfy the multidimensional conditions on *Q*. The second group contains the *NoSet* where at least one field does not meet all the conditions on different fields. Further for each record will

also calculate the *rowhash* in order to authenticate the content of the entire view for each item: (key, rowhash). See table 3.11.

Cod.	Student	Homework	Grade	Set	Rowhash
530	White Mark	NULL	C	YESset	$h(r_4)$
501	Smith Joe	NULL	NULL	NOset	$h(r_6)$
525	Cornwell Sharon	NULL	NULL	NOset	$h(r_8)$
489	Moore Mark	NULL	NULL	NOset	$h(r_{10})$

Table 3.11: *Step 3*: tuples that result from query 3.1 and 3.2 that belong to *YESset*. The other are tagged with *NOset*. Further we calculate the *rowhash* for each record.

The advantage of this approach is the possibility to perform concurrent queries on the *scores* table (in an unauthenticated way) and then validate it using only one authenticated skiplist. The Step 4 validates the query result. The validation uses only the first column skiplist returned. A column is sufficient to validate the entire table, because is possible to verify:

- the *integrity* for each record of table *scores* using just a single field of it, because each node of the authenticated skiplist contains a pair  $(x, h(r_x))$  where  $x$  is the value in the table field and  $h(r_x)$  is the *rowhash* of the entire record, that is the hash of all the fields belonging to it
- the *completeness* of the result of  $Q$  because we verify the *FCR* that is a superset of the final result and then we filter with the conditions on the other fields

If the validation is successful then the *YesSet* shows the result of multidimensional query  $Q$ . The result of the query is visible in the table

## Join

The *join* is one of the most problematic relational primitives to authenticate, the main problem is to authenticate the completeness at the best of our knowledge the only solution available in literature [61] is to perform the Cartesian product of the entire tables and then to verify that the join result is contained in it.

To perform an authenticated join between two authenticate tables we use the merge scan technique [92] that is a standard in modern DBMS because we exploit the necessary order to maintain an authenticated skiplist. The two

### 3.4. OUR APPROACH

43

input tables create two independent processes. Each one of these two processes authenticate the join table using the algorithm *FCR*. For instance we use the Query 4 in section 3.2 that is divided in two sub-queries, one for each table. We use algorithm *FCR*.

- `SELECT *`  
`FROM Jscore`  
`WHERE Grade > 'C'`
- `SELECT *`  
`FROM Students`

The output of this step are two authentication tables. The join between two tables authenticate is realized using the mergescan join algorithm. Since both of these tables are ordered following the join attribute, all the conditions for the merge scan execution are met.

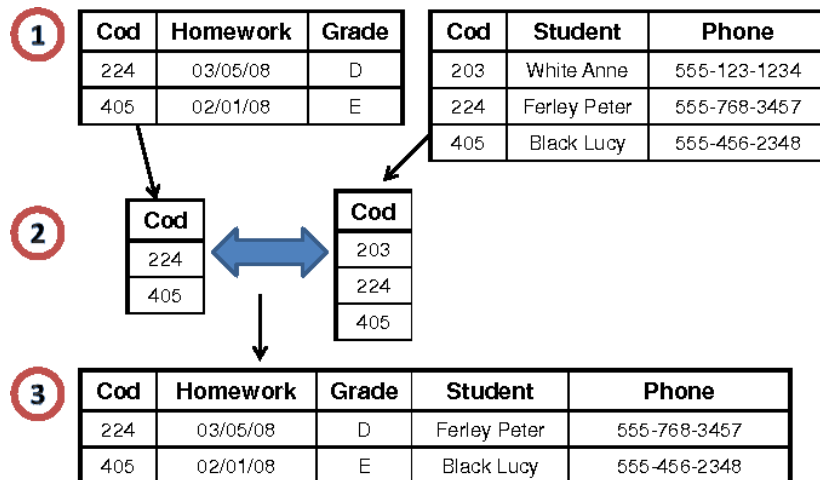


Figure 3.3: *Authenticated Join Algorithm.*

Fig. 3.3 shows the comparison process among the elements of the two tables. During the concurrent scan we compare two tuples searching for pairs

that match. The concurrent scan proceeds leaping from a table to the other, looking for elements equal, in detail you can see the following algorithm.

```
while (elemA != NULL or elemB != NULL)
{
    if (elemA = elemB) {
        createJoinTuple(elemA, elemB);
        elemA = elemA.next;
        elemB = elemB.next;
    }
    else if (elemA < elemB) {
        while (elemA < elemB)
            elemA = elemA.next;
    }
    else {
        while (elemB < elemA)
            elemB = elemB.next;
    }
}
where elemA and elemB are element of the tables.
```

## Set Operations

The *Set Operations* for this section is considered as operations on different fields of the same table.

### Union

operator is managed as a merge between query results from different conditions on more fields of the same table. Each condition belongs to a different task, which can be executed in parallel. Each condition is transformed in a simple select query so we can exploit the same method showed for the standard select operation. So we check integrity and completeness for each component of the *union* operator and the merge among all *yesSet* of each condition is the result set for *union*. In fact only if all queries finish successful, the integrity will be guaranteed.

### 3.4. OUR APPROACH

45

#### Intersection

operator is managed as we managed the select operation. Therefore, intersection between two different condition is the same of a select with two conditions.

#### Difference

operator is based on the select query process, too. We apply the algorithm proposed for the *select* operation directly on the first member of the difference operation. The output of that operation is the *yesSet* in which the client can filter the elements that belong the first condition and does not belong to the others. We use the same *multidimensional authentication algorithm* to ensure integrity and completeness properties.

#### Algebraic Operations

The proposed algorithm allows to authenticate only *algebraic operations* that do not need aggregation and in articular we can verify *min* or *max*.

#### Min and Max

operators can be implemented by our proposed approach by exploiting the ordered requirement of *ADS*. Therefore, we can know which is the first element or the last element in a set of data. *ADSeS* maintain a sorted set of data and it is possible to use any order, that it is to say if we need to know the first or last element of a set that is ordered in two different ways we need two different ordered *ADSeS*. So retrieve *min* or *max* element over a range is very easy. We can distinguish three cases of *min* or *max*:

- only one condition on the same attribute of *min* or *max*.
- more conditions on the same attribute of *min* or *max*.
- more conditions except the attribute of *min* or *max*.

The first case can be managed as a simple select query, because there are no more conditions and the *yesSet* is already sorted. So we get the first element for *min* (the last for *max*). The second case is analogous to the first case. There are some conditions, but there is a condition expresses over the column which belong the min (max) attribute. So, according to the difference operator, we force *FCR*: the task with the min condition will be the first task to finish.

The related *yesSet* belongs to this column and then we retrieve first element for min (last for max). The third case is more complex, it requests two skiplists to assure integrity and completeness. The result of the select query, will match the *yesSet* through the *min* or *max* condition and then retrieve the result. Then this value must be tested on the related skiplist to ensure the integrity and completeness of this untrusted operation.

### Range optimization

To validate a set of data in a contiguous range a straightforward solution is to validate every set element individually. In this way we do not exploit the order of the element. In fact we can exploit the authentication semi-path as for example on elements 6, 12, 13 and 16 in Fig. 3.4.

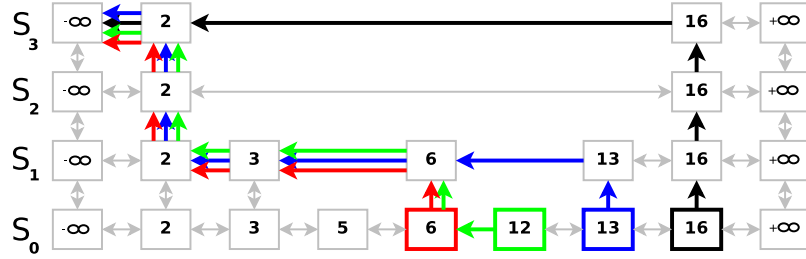


Figure 3.4: Authentication shared paths. The four elements 6, 12, 13 and 16 are contiguous. So, if we consider them individually we obtain different paths (the one way arrows) but if we exploit that these elements are contiguous we can notice that they have several intermediate nodes in common (starting from element 6 at level  $S_1$  we have three paths together).

In details the skiplist element in position  $v$  has an authentication path that arrives up to *basis* that is at the top level  $(-\infty, S_3)$ . See Fig. 3.4. For a node in position  $v$  the authentication path starting from node in position  $v$  is partially overlapped with the path of the node in position  $v - 1$ . In a range of elements starting from position  $v_{min}$  to position  $v_{max}$ , the authentication path of all elements is completely inside the range, that we call the *result path*. We observe that all nodes in the range are partially overlapped with the path of the greatest node in position  $v_{max}$ . Therefore, the *result path* is mainly overlapped with the path of the greatest node in position  $v_{max}$ . The

### 3.5. EXPERIMENTAL EVALUATION

47

remaining nodes contribute to the *result path*, by adding the missing parts. That is, the authentication path of node in position  $v_{max}$  is the *backbone path* and the previous nodes contribute for the semi-paths in the range  $[v_{min}, v_{max}]$ . Each element in the *backbone path* we call *pivot*, for instance, referring to the fig. 3.4, if the element with label 13 is in position  $v_{max}$ , each element in its authentication path with label: 13,6,3,2, $-\infty$ , is a *pivot*. In order to find all semi-paths in the range, we divide the *result path* in some *areas*, each bounded by two *pivots*, that we locate in the *backbone path*. See fig. 3.5. Each *area* contains elements which are authenticated using the techniques described below. Our approach exploits the authenticated skiplist as described in [34] in which any skiplist element authentication path refers to the *basis*. The main idea is to define for any *pivot* a list of new *local basis*. This list evolves during each iteration of range algorithm, as we show in Fig. 3.5. For each iteration we authenticate the first element next to the left *basis-list*. After authentication we add this authenticated element in the *local basis-list* and we continue until we authenticate the element at the right *basis-list*. This algorithm allows to shorten at every step the authentication path of the element to authenticate.

## 3.5 Experimental Evaluation

### Setup

This section shows the experimental results obtained using a prototype implementation of the techniques presented in the previous sections.

The Hardware architecture where tests have been performed consists of a standard laptop with following features:

- cpu Intel(R) Core(TM)2 Duo CPU T8100 2.10GHz
- RAM 4 Gb DDR2
- HDD 5,400 rpm Serial ATA

The Software architecture consists of following elements:

- Linux<sup>TM</sup>2.6.24-16-generic on Ubuntu<sup>TM</sup>8.04 (hardy) 32bit;
- Java<sup>TM</sup>JDK version 1.6.0 update 6
- MySql JDBC Connector Java-bean 5.1.15

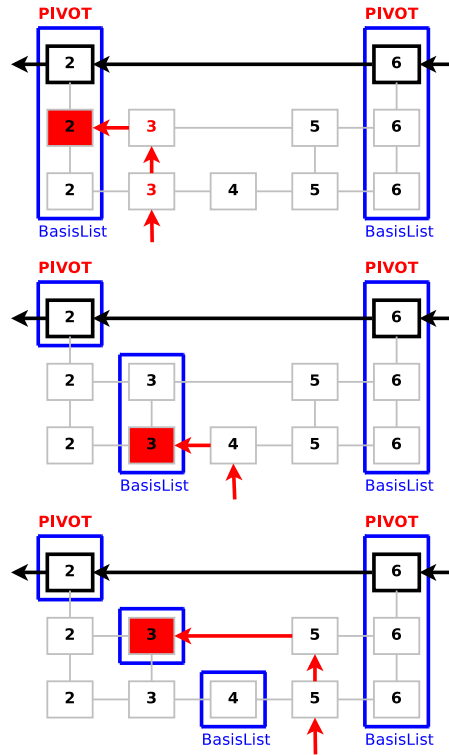


Figure 3.5: Basis-list update. In the first step the *basis-list* of the node 3 is the node 2 at level 1. In the second step the *basis-list* of the node 4 is the node 3 at level 0. In the third step the *basis-list* of the node 5 is the node 3 at level 1. The algorithm stops when we reach the next *pivot* element 6.

### 3.5. EXPERIMENTAL EVALUATION

49

- MySql DBMS version 5.0.45
- NetBeans IDE 6.0 (Build 200711261600)

The JAVA virtual machine was launched with **-Xmx1500m** parameters, that is 1,5 Giga Bytes of RAM. Furthermore, all tests used NetBeans during normal computer operations. For the setup of these experiments we used two different sets of data.

- The first group is an artificial set of data randomly built. We created different tables with the number of records that ranges from 10,000 to 1,000,000 of elements and with the numbers of columns that ranges from 1 to 100 fields. To raise the randomness of the tests we decided to rebuild the entire set of the test table. Also the authenticated data structures were computed each time starting from a different main table. All the elements contained into the table have the type *string*.
- The second group is a set of real data, made freely available online by the University of Irvine (California) for testing on Data Mining and Machine Learning [4]. A detailed description of this set is given in section 3.5.

All values presented in this section have been computed as average of the results of 10 different tests. All times are in *milliseconds*. All tests show the *clock-wall* time.

#### Case study on artificial data set

In this section we show some example queries to authenticate. We execute on an artificial random data set. The data type for all data stored on the database is String (VARCHAR). The results of artificial data set experiments range from Table 12 to Table 19.

	1	10	100
insert	63	439	4373
delete	63	426	5748
select	42	44	60

Table 3.12: Queries performed on a single table element with 10,000 records with the columns that range from 1 to 100.

	10,000	100,000	1,000,000
insert	439	3108	7387
delete	426	3066	7532
select	42	54	65

Table 3.13: Queries performed on a single table element with 10 columns with the records that range from 10,000 to 1,000,000.

	1	10	100
select	556	629	3847

Table 3.14: Queries performed for all elements belonging to a single column to authenticate the entire table with a variable number of fields. In a table with 10,000 records and with the fields that range from 1 to 100.

	10,000	100,000	1,000,000
select	626	7910	195039

Table 3.15: Queries performed for all elements belonging to a single column to authenticate the entire table with a variable number of records. In a table with 10 fields and with the records that range from 10,000 to 1,000,000.

### Case Study on Real Data

In this section we propose authenticated queries performed on real data and no more on artificial data as in the previous section.

This set is publicly available from University of California Irvine, through the *UCI Machine Learning Repository* [4]. We have chosen for our test the *Adult Data Set*<sup>1</sup>. The extraction of the original data was made by Barry Becker [4] from the 1994 Census database, where he filtered some information from the database, for instance he removed the record where the age is less than 16 years. The data contained in the database Adult are divided in two categories of people who earn more than 50,000\$ a year and those who earn

<sup>1</sup>This database is available at this address <http://archive.ics.uci.edu/ml/datasets/Adult> Data are stored in file *adult.data*

### 3.5. EXPERIMENTAL EVALUATION

51

	1	10	100
join AUTH	4250	4390	22802
join NOT AUTH	3696	4160	13438
join CART	OutMem	OutMem	OutMem

Table 3.16: Self Join Query to authenticate the table with 10,000 records and with the number of columns that ranges from 1 to 100. The line *join AUTH* shows the join authentication time. The line *join NOT AUTH* shows the time that takes the same query performed with a classic join, without checking value integrity and completeness in the ADSes structures. The line *join CART* shows the join that we implemented as a Cross join (Cartesian product of two tables) the only other technique available in literature. This type of implementation is faster than other two type, only with a few element. For a table with only 10,000 element Join Prod required more than 2,0GB of RAM, so we have an Out Of Memory Exception thrown by Java virtual machine. We show the gap between join AUTH and join NOT AUTH is of the same size.

Table A	Table B	Auth	Not Auth
10,000*10	10,000*10	-	-
Tuples 976		1833	1315

Table 3.17: We show the gap between join AUTH query and join NOT AUTH query over two different tables with the same size: 10,000 records and 10 columns.

Table A	Table B	Auth	Not Auth
100,000*10	100,000*10	-	-
Tuples 9425		162897	159675

Table 3.18: We show the gap between join AUTH query and join NOT AUTH query over two different tables with the same size: 100,000 records and 10 columns.

less.

The first group of people represents 23.93% of the total, those of the second

Table A	Table B	Auth	Not Auth
10,000*10	100,000*10	-	-
Tuples 936		38921	24290

Table 3.19: We show the gap between join AUTH query and join NOT AUTH query over two different table with size: the first has 10,000 records and 10 columns, the second has 100,000 records and 10 columns

are the 76.07%. People of both groups have an age that ranges from 17 to 90 years. The table has a size of 15 columns for about 32,000 records.

The *fields* describe for each person: age, workclass, education, marital-status, occupation, sex, hours for week, native-country, capital gain, etc.

### Query 1

Determine how many young people who earn more than 50k\$?

```
SELECT *
FROM adult
WHERE makemoney = '>50K'
AND age BETWEEN '17' AND '25';
```

returned records	114
involved columns	2
authentication time	1363 ms
records in the view	6411

### Query 2

Determine how many young people of “Amer-Indian-Eskimo” race that earn more than 50k\$?

```
SELECT *
FROM adult
WHERE makemoney = '>50K'
AND age BETWEEN '17' AND '25'
```

### 3.6. CONCLUSIONS

53

AND race = 'Amer-Indian-Eskimo';

returned records	2
involved columns	3
authentication time	392 ms
records in the view	311

#### Query 3

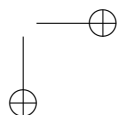
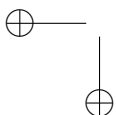
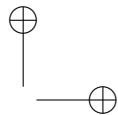
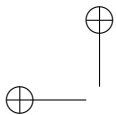
Determine how many men, born in the U.S., earn more than 50k and work in the private sector?

```
SELECT *
FROM adult
WHERE makemoney = '>50K'
AND workclass = 'Private'
AND nativecountry = 'United-States'
AND sex = 'Male';
```

returned records	3879
involved columns	4
authentication time	14543 ms
records in the view	7841

### 3.6 Conclusions

In this chapter we presented an extension of the techniques in [23] to authenticate the integrity and completeness of query results on relational tables with conditions on different fields at the same time. The method exploits concurrent processing techniques that allow to bind the complexity to the most selective field in the query. This approach allows to authenticate efficiently the relational join operation.



## Chapter 4

# Efficient Integrity Checking of Untrusted Network Storage

### 4.1 Introduction

Outsourced storage has become more and more practical in recent years. Users can now store large amounts of data in multiple servers at a relatively low price. An important issue for outsourced storage systems is to design an efficient scheme to assure users that their data stored in remote servers has not been tampered with. This chapter presents a general method and a practical prototype application for verifying the integrity of files in an untrusted network storage service. The verification process is managed by an application running in a trusted environment (typically on the client) that stores just one cryptographic hash value of constant size, corresponding to the “digest” of an authenticated data structure. The proposed integrity verification service can work with any storage service since it is transparent to the storage technology used. Experimental results show that our integrity verification method is efficient and practical for network storage systems. In this chapter, we propose an efficient and secure technique that allows the client to verify the integrity and completeness of network storage without having to trust the network storage system.

### Our Contributions

The main contributions of this chapter are the following:

1. We propose an architecture for verifying the integrity of untrusted outsourced storage. For our method to work, no trust is needed at either the storage server or the authentication server (see the definitions above). Our integrity verification service is independent from the storage service and works with any existing storage technology. Note that our solution addresses only the problem of integrity checking. Other security services, e.g., user authentication and data encryption, are orthogonal to and compatible with our service and are not addressed in this dissertation.
2. We provide efficient algorithms and protocols (of logarithmic complexity) for checking the integrity of data stored at an untrusted storage server using only  $O(1)$  space at the client. Namely, suppose that the storage server keeps a file system with  $n$  files. The client can verify the integrity of a file downloaded from the storage server in  $O(\log n)$  time. Also, the client can verify the correctness and completeness of the list of  $k$  file names matching a given path prefix returned by the storage server in  $O(k + \log n)$  time.
3. We implement a prototype of our integrity verification system that works with Amazon’s *Simple Storage Service* (S3) [1].
4. We present the results of experiments on the performance of our prototype, focusing on the communication and processing overhead incurred on top that of Amazon S3. The experiments show that our system provides integrity checking while adding minimal overhead to the normal operations of Amazon S3.

Our architecture has several advantages over many previous methods. Our system requires only constant amount of storage (a single cryptographic hash value) on the client side, irrespective of the amount of outsourced data. Integrity checking is achieved efficiently, with virtually no observable overhead for file systems with hundreds of thousands of files. We maintain authentication information using an authenticated skip list (see Section 1.3), which supports simple and fast updates. Unlike some of the previous approaches, the security of our scheme is independent from probabilistic assumptions about the extent of data corruption. Instead, our system is as secure as the cryptographic hash function used. We do not assume that any component of either the storage server or the authentication server is trusted, therefore any attack on either server will be detected, even if the two collude in an attack. Thus, the authentication server itself can be an outsourced computational resource.

## 4.2. OUR APPROACH

57

Another major characteristic of our architecture is that it operates in the single-client setting, unlike other approaches such as SUNDRA [55] which supports an authenticated file system in a multi-client setting, but achieves a weaker notion of consistency. This form of consistency is called *fork-consistency* and disallows anything more than the forking attack, where two clients can have a different view of the file system. In our case, full consistency of the file system in a multi-client setting can be provided either by serializing operations from different clients through a common trusted client (e.g., this can be the kernel of the file system), or by requiring each client to communicate its fresh state to all other clients after an update. The latter approach requires additional  $\Omega(c)$  communication for  $c$  clients.

Our model also differs from data retrievability models such as PDP [5] in a number of ways. Our goal is not to detect corruption of data stored on the server, but to verify that the server’s *responses* to the client’s queries are consistent with the updates that the client has performed in the past. Thus, integrity checks are performed only when a file or list of files is requested from the storage server. The full response can then be used to verify integrity. Also, we do not require the client to keep any secret information such as a private key, an important distinction in situations where users would like to collaborate without fully trusting each other. Additionally, we are able to verify the completeness and correctness of lists returned from the server as well as the data itself. Finally, no cooperation between the client and storage server beyond the normal, unauthenticated case, is necessary. As a consequence, our integrity checking system can sit on top any existing storage service without the knowledge and cooperation of the storage server.

## 4.2 Our Approach

We present a method that allows the client to manage and verify the integrity of content hosted on a remote storage server. Our method uses only a small, constant amount of storage on the client’s computer, while the rest of the data needed for integrity checking is hosted on a separate authentication server (see Figure 4.1). Our technique assumes that both the storage server and the authentication server are untrusted. We can detect any data corruption on either server, even if the two cooperate in an attack. Our authentication server stores authentication information in an authenticated skip list, a data structure described in Section 1.3 that supports efficient updates and queries.

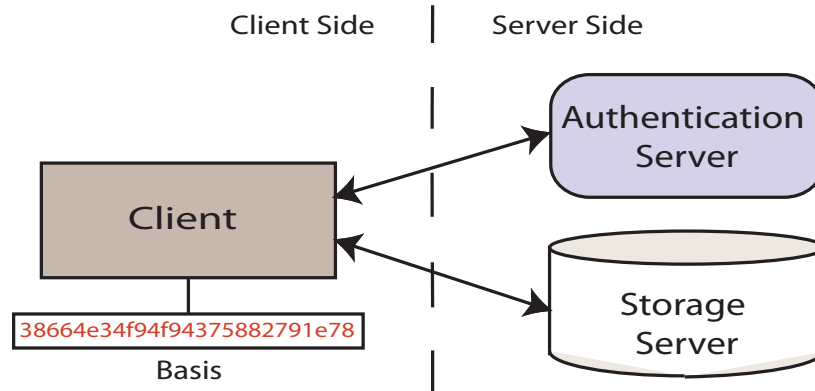


Figure 4.1: Reference model: The client stores only one hash value (the basis) to verify the integrity of all content on the network storage system.

### Problem Definition

The problem we address involves two parties: an untrusted server component consisting of the storage and authentication servers, and a client. Even if the standard user identification scheme (Kerberos, for example) used by the storage server protects the client’s data from outside attackers, there is still the possibility of a threat from an attacker within the storage server, for example from someone that has unrestricted access to the client’s authentication information and account. How can the client be assured that his data will not be tampered with? We need to be able to detect such tampering in the following cases:

- The client requests a list of all of the objects with a given prefix that have been stored in the server, and the response is incomplete or incorrect.
- The client downloads an object from the untrusted storage server, and the content of that object has changed since the client uploaded it.

As an artifact of our architecture, we additionally must detect the case where an operation requiring authentication is performed (a list, download, upload, or deletion) and the portion of the data stored on the authentication server that is needed to authenticate the operation has been corrupted.

## 4.2. OUR APPROACH

59

### General Architecture

We have designed a general object-oriented software architecture for authenticated network storage services and we have implemented it in Java. A high-level view of the software architecture is shown in Figure 2.1. In our architecture, there are three entities, the first two of which reside on the server side, and the last of which resides on the client side:

- The *storage server*, which can be any storage service available online. The storage server is untrusted.
- The *authentication server*, which manages all of the authentication information. We run software on this server which is capable of building and maintaining an authenticated skip list structure in response to update requests received from the client, as well as responding to the client's queries about the integrity of outsourced data with proofs of authenticity or corruption. A proof consists of an ordered collection of hashes (a hash chain) and some information about the structure of the authenticated skip list. The authentication server is also untrusted.
- The *client*, who can query both the storage and authentication servers remotely and verifies the answers given to it. Verification is achieved through comparisons to a hash value stored by the client, the basis of the authenticated skip list on the authentication server. This hash (along with the software itself) is the only data which must be stored on the client, and it has constant size dependent only on the cryptographic hash function used. We assume that data stored, and operations performed on the client are entirely trusted, and as this hash value is computed and stored directly by the user when he performs an update, it is trusted. In fact, it is the only trusted value in the entire proposed solution. We run software on the client that makes use of the authentication server to authenticate the client's queries to the storage server. It is worth noting that in the most general case, this authentication software will simply provide an interface that any unauthenticated system can plug in to. Such an API has not been implemented as of now, however, and the implementation presented in this section is more specific to the particular storage service used.

To illustrate how this architecture functions, we describe the sequences of actions triggered by some common user requests. Suppose that a user

would like to store a file in the storage server and wants to authenticate the PUT operation (see Figure 4.2). The following steps are performed:

1. The user selects the file to upload
2. Our client side software sends two different update queries, one to the storage server and the other to the authentication server.
  - The storage server query adds the user’s file to the server.
  - The authentication server query, which contains the hash of the file, updates the authenticated skip list on the server and retrieves a proof which allows the client to compute the correct new basis.

At a later time, the user would like to retrieve the file and wants to authenticate the GET operation (see Figure 4.3). Then the following steps are performed:

1. The user selects the file to download.
2. Our client side software sends two different queries, one to the storage server and the other to the authentication server.
  - The storage server query retrieves the user’s file.
  - The authentication server query retrieves the proof of integrity
3. When the client receives both answers, it can verify the integrity of the file (see more details in the next section).

### Algorithms and Complexity

In this section we describe the technical details of our architecture. Suppose a client stores  $n$  files (in fact, keyed data blocks of any size can be used) in a storage server, and maintains a corresponding authenticated skip list structure (refer to Section 1.3) at an authentication server. For each file in the storage server  $(k_i, f_i)$ , a tuple  $(k_i, h(f_i))$  is stored in the skip list, where  $k_i$  is the key (name) of the file with content  $f_i$ , and  $h(f_i)$  is a cryptographic hash of  $f_i$ . The storage server and the authentication server are synchronized so that they contain the same elements. The basis of the authenticated skip list is stored locally by the client. The client now can issue four main operations which we describe and analyze below. For each of these operations, the main measures of complexity that we are interested in are the following:

#### 4.2. OUR APPROACH

61

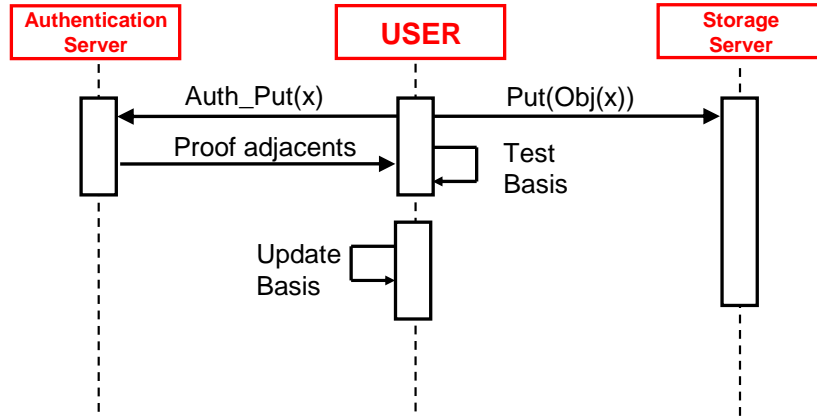


Figure 4.2: Authenticated PUT for a file with key =  $x$ .

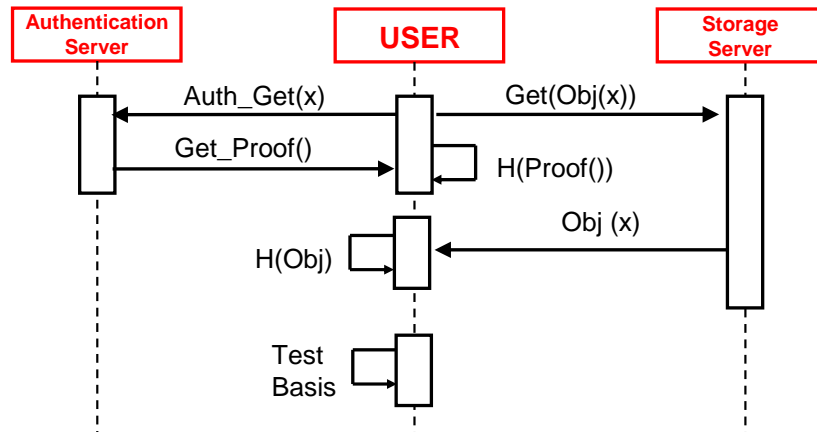


Figure 4.3: Authenticated GET for a file with key =  $x$ .

1. **Query Complexity.** The time needed for the authentication server to construct the proof in response to a query (either a GET or a LIST or a PUT or a DELETE query).
2. **Verification Complexity.** The time needed for the client to process the proof in order either to verify a GET or LIST query or to update the

basis after a PUT or a DELETE query.

3. **Update Complexity.** The time needed for the authentication server to perform an update (either an authenticated PUT or an authenticated DELETE).
4. **Communication Complexity.** The size of the proof (previously referred to as  $p$  or  $p'$ ) that must be sent over the network in response to a query.
5. **Hashing Complexity.** The number of hash computations executed during a verification or an update.

**Authenticated PUT( $k, f$ )** The client wants to upload to the storage server a file  $f$  named  $k$ . He sends the request to both servers. The authentication server adds a new entry  $k$  associated with the element  $h(f)$ . At that point it also sends a proof  $p'$  back to the client (see Figure 4.2). In this case  $p'$  contains information that allows the client to compute the new basis. Referring to Figure 1.4, one can see that if we insert a node with  $k = \text{"D"}$ , the only nodes in the skip list whose hash values will change are the rounded ones - the search path for  $k$ . To recompute the hash values of these nodes, we need only the hash values of the nodes bordering this search path (nodes whose arrows end at a shaded node), therefore, the proof will contain those hash values. The length of the search path is  $\log n$  with high probability (w.h.p). It follows that the complexity of this operation in all five of the above categories is  $O(\log n)$  with high probability. Before computing the new basis, the client validates the proof against the current basis. In this way the client is assured that the new basis he computes is correct.

**Authenticated DELETE( $k$ )** The client wants to delete a file from the storage server with name  $k$ . This procedure is similar to the procedure PUT( $k, f$ ). The complexity of this operation is also  $O(\log n)$  with high probability.

**Authenticated GET( $k$ )** The client wants to retrieve from the storage server the file contents of the file with name  $k$ . The hash of the file  $h(f)$  is stored at a leaf of the authenticated skip list on the authentication server. The client makes a query to the authentication server that returns  $h(f)$  along with a proof of the integrity of  $h(f)$ . Once again, this proof consists of information from the nodes bordering the search path, so the complexity of the operation in each of

## 4.2. OUR APPROACH

63

the applicable categories is also  $O(\log n)$  with high probability. The proof can be verified against the client’s stored basis, and if the verification succeeds, the client can check to see that the hash of the file received from the storage server equals  $h(f)$  (see Figure 4.3).

**Authenticated LIST(prefix)** The client wants to retrieve the names (but not the contents) of all the files whose name begins with **prefix**. We have developed a method for efficiently authenticating a list of  $k$  elements taken from a server containing  $n$  elements. We obtain a proof from the authentication server that includes the hashes of each of the  $k$  elements (the list body), and parts of the proofs for **GET** operations performed on the **prefix** and the last list body element (see Figure 1.4). Additionally, the proof contains the heights of the towers associated with each of the above nodes. We will show that the query and communication, hashing, and verification complexity of this operation is  $O(k + \log n)$  with high probability.

To determine the construction time, we assume that the only time-relevant operation is a comparison, and that this operation takes  $O(1)$  time. Referring to Figure 1.4, one can see that the proof for a **LIST** operation includes elements from the proofs for **GET** operations on the **prefix** and the last element in the list. The number of comparisons performed on the server for a **GET** operation is  $O(\log n)$  (the height of the skip list). Additionally, the proof contains information about each of the  $k$  elements making up the body of the list. The query we make for the list body portion of the proof has two steps. First we search for the **prefix** - this is  $O(\log n)$  as well. Second, we move to the right until we reach the end of the list - an additional  $O(k)$  comparisons. Summing all of the portions of the proof construction process, we see that the number of comparisons (the query time for **LIST** operation) is  $O(k + \log n)$ . As a result, the size of the proof must also be  $O(k + \log n)$ , since the size of the proof cannot exceed the number of elements considered during its construction.

After the proof for a **LIST** query has been built and sent to the client, the client has to run a verification algorithm in order to recompute the basis (which he maintains locally) from the proof. We start with a pointer to the rightmost proof element and maintain a stack  $S$  of proof elements as we proceed to the left. While the height of the current proof element is greater than or equal to that of the stack top, we pop the stack top and absorb its hash value into the current element using a commutative cryptographic hash function. Otherwise, we push the current element onto  $S$ , and move the pointer to the left. This verification algorithm processes a proof of size  $O(k + \log n)$ , and one can see

that each element of the proof is passed in to the hash function exactly once. Since the computation of the hash function takes  $O(1)$  time, it follows that the verification algorithm takes time  $O(k + \log n)$ .

### Security

Our service provides protection against a wide range of attacks. An attacker may gain access to our storage server and damage or delete some of our files, or gain access to our authentication server and alter some authentication data, or do both simultaneously in order to try to deceive the client. An attacker may also intercept network communication from the client to one or both servers and change the message contents. The computations performed on the authentication server to update the authenticated skip list may also be controlled by an attacker, resulting in corrupted authentication information. In this section we will show that as long as the client itself is not compromised and the attacker is computationally bounded, the probability that *any* attack on the untrusted portion of the service will not be detected is negligible (see definition of negligible function below).

Here we give a definition of security for our protocol. We recall that a negligible function  $\nu(k)$  is a function that decreases faster than any inverse polynomial  $p(k)$  as  $k$  increases ( $k$  is the security parameter, in our case the length of the output of the collision-resistance/cryptographic hash function we use). We also recall that for the specific cryptographic primitive we use, i.e., the collision-resistant hash function, the probability that a computationally bounded adversary can find a collision is  $\nu(k)$ .

**Definition 1 (Security)** *Given a storage server  $S$ , an authentication server  $A$  and a client  $C$  that stores  $n$  files on  $S$ , we say that an integrity checking protocol is secure if:*

- *For a file  $f'$  named  $x$  stored in  $S$ , the probability is negligible that after a  $\text{GET}(x)$  query,  $A$  computes a proof  $p$  and  $S$  sends  $f'$  such that  $(p, f')$  passes the verification test, when in fact, the data in  $f$  is corrupted.*
- *For a list  $Y'$  of names with prefix  $y$  of files stored in  $S$ , the probability is negligible that after a  $\text{LIST}(y)$  query,  $A$  computes a proof  $p$  and  $S$  sends  $Y'$  such that  $(p, Y')$  passes the verification test, when in fact,  $Y'$  is either incorrect or incomplete.*

We can now prove that our protocol is secure according to Definition 1. Suppose in the beginning (when the data structure contains one element, for

#### 4.2. OUR APPROACH

65

example) the client possesses the correct basis. Suppose he issues a  $\text{GET}(x)$  query. The server needs to hide the fact that it has tampered with the data of the file named  $x$ . In order for the server to do that, it must either find another file  $f'$  that has the same hash as the original file and send the correct hash and the incorrect file, or find another hash (for the incorrect file  $f'$ ) that will produce the same basis if included in the hashing scheme of the skip list. Neither task can be accomplished with non-negligible probability since both require finding a collision in a collision resistant hash function - in the former case, the function used to store the files in the leaves of the skip list, and in the latter case, the function used for the hashing scheme within the skip list. This argument can be applied for the  $\text{LIST}$  query as well.

However, the above is true only if the client always maintains the correct basis, even after updates take place. Indeed, for every update (either  $\text{PUT}$  or  $\text{DELETE}$ ) the client runs an algorithm that takes as input the proof  $p'$  created by the authentication server, some necessary structural information which is included in the hashing scheme, and the existing basis, and outputs the new basis corresponding to the correct authenticated data structure after the update has taken place (See Figure 4.2). This technique ensures that the basis stored by the client is equal to the hash of the head node of the correct authenticated skip list at all times. One important result is that if an attack is made on the authentication server, altering the skip list stored there, the client will know, because the client's basis corresponds to the *correct* skip list, and the one on the server is now incorrect. This and other practical examples of attacks are discussed below.

Unlike some other security schemes that detect data corruption with some variable uncertainty [5], which basically solve a different problem, our approach guarantees that such corruption will always be detected (negligible uncertainty). We accomplish this high level of security by maintaining the correct basis on the only trusted component of the system, the client. When an update is made and the basis needs to be changed, all of the relevant computations are also performed on the client, and their correctness is verified against the old basis. In this way, we ensure that the basis will be updated correctly on the client, even if update operation on the server is compromised by an attacker. The possession of this basis allows us to protect against all of the types of attacks mentioned earlier. Even if there is some malicious cooperation between the authentication and storage servers, the attack will be detected - either the proof provided by the authentication server will not agree with the data from the storage server, or it will not agree with the trusted basis on the client, and in either case the client will know there is a problem. Also, note

that from the client’s perspective the cases for which an attacker intercepts and alters network traffic between client and server are identical to those for which the actual data stored on the servers is altered, therefore our security model is equally adept at detecting them. It is worth pointing out that once we detect an attack, we will not always be able to determine *which* portion of the system was attacked. If an attacker manages to alter some data on the authentication server, the server may not be able to provide a correct proof of integrity to the client, and the client will be unable to determine whether or not an attack on the storage server has occurred as well. For clarity, we summarize these concepts by distinguishing the following cases:

1. No attack is made on either the authentication or storage servers. Result: The client can verify that integrity is preserved.
2. An attack is made on the storage server, but not the authentication server. Result: the attack is detected, and the client determines that the integrity of the data on the storage server has been compromised.
3. An attack is made on the authentication server. Result: the attack is detected, but it may not be possible for the client to determine whether or not the data on the storage server has been corrupted as well.

From a practical perspective, we view the authentication server and the storage server as a single untrusted entity, and although it would be useful to be able to determine the status of the data on the storage server even if the authentication server has been attacked, the only crucial point is that the probability that any attack on the untrusted portion of the service will not be detected is negligible. The only practical disadvantage of separating the untrusted authentication and storage components is two servers instead of one are exposed to attacks. The security of the servers themselves, however, is a topic outside the scope of this work.

Based on the efficiency of the skip list data structure (main operations run in expected time  $O(\log n)$  with high probability (w.h.p)), the results for the LIST implementation we derived before, and the proof of security above, we can summarize the main complexity and security results of this section:

**Theorem 1** *Assume the existence of a collision-resistant hash function. The presented protocol for checking the integrity of  $n$  files that reside on the storage server supports authenticated updates PUT() and DELETE() and authenticated queries GET() and LIST() and has the following properties:*

### 4.3. IMPLEMENTATION

67

1. *The protocol is secure according to Definition 1;*
2. *The expected running time, communication complexity and hashing complexity of PUT(), DELETE() and GET() is  $O(\log n)$  at the server and at the client with high probability;*
3. *The expected running time, communication complexity and hashing complexity of LIST() is  $O(k + \log n)$  at the server and at the client with high probability, where  $k$  is the size of the returned list;*
4. *The client uses space  $O(1)$ ; and*
5. *The server uses expected space  $O(n)$  with high probability.*

Taking into account constant factors (see the definitions in [100]), the communication and hashing complexity can be shown to have an upper bound with high probability of  $1.5 \log n$ .

### 4.3 Implementation

To validate our software architecture for online storage authentication, we have implemented a prototype of an authenticated network storage service. Our prototype utilizes three pre-existing services/applications: Amazon Simple Storage Service is the untrusted data storage server, Amazon Elastic Compute Cloud provides our untrusted authentication server, and the prototype is built on top of an existing open source project called Jets3t Cockpit. In this section, we present some details about these three components, and then proceed to discuss the architecture of our implementation.

#### Amazon S3 and EC2

Amazon Simple Storage Service (S3) is a scalable, pay-per-use online storage service. Clients can store a virtually unlimited amount of data, paying for only the storage space and bandwidth that they use, with no initial start-up fee. The basic data unit in S3 is an object. Objects contain both data and meta-data. Only the meta-data portion is used by S3. The basic container for objects in S3 is called a bucket. Buckets are flat, as opposed to hierarchical; they cannot contain other buckets, only data in the form of objects. Each bucket in S3 has a unique name, and each object has a key that identifies the object within its bucket. A single object has a size limit of 5 GB, but there is no limit on the

number of objects per bucket. Each client is limited to 100 buckets. Despite the flat storage scheme, it is possible to simulate hierarchical relationships through either special naming conventions (use of “/” or “.” to denote directories) or use of customized object meta-data (pointers to associated files, for example). S3 supports both SOAP and REST requests.

Amazon Elastic Compute Cloud (EC2) is a pay-per-use service that provides online computing resources. A client can start a virtual machine (instance) on EC2 using any complete image of a machine. EC2 makes a number of public images available for running servers, database management systems, development environments, and so on. Clients can also run customized images.

### Jets3t Cockpit

Cockpit is a subset of the open source project Jets3t. It is written in Java. It provides a graphical front-end for managing content stored on S3. The original functionality of Cockpit included support for LIST (with the option of specifying a prefix and/or delimiter) and download (GET) queries, as well as upload (PUT) and delete (DELETE) operations. Additionally, the software provides optional encryption of uploaded data and more advanced features such as generation of public URLs that allow general access to a bucket in S3 for a limited time [2].

### Software Architecture

We have added integrity checking to the four basic operations of Amazon S3: the LIST, GET, PUT and DELETE. Note that these four operations form the core of any storage service. When the client triggers one of these operations, a new call is made in parallel with the original call to S3 (which is left unchanged), to an integrity checker that talks to EC2, where our authentication server resides (see Figure 4.4). The GUI of Cockpit has been modified to accommodate the additional authentication information.

An abstract class `IntegrityChecker` (see Figure 4.5) provides the template for any integrity-checking service. It specifies four abstract methods, corresponding to the authentication of LIST, GET, PUT and DELETE operations, which must be implemented by any child class. Currently, the only implementing class is `STMSIntegrityService`, which delegates the authentication tasks to a service that stores and retrieves authentication data in main memory on an EC2 instance through Java Remote Method Invocation (RMI). Another service based on a database management system (DBMS) is in development and

### 4.3. IMPLEMENTATION

69

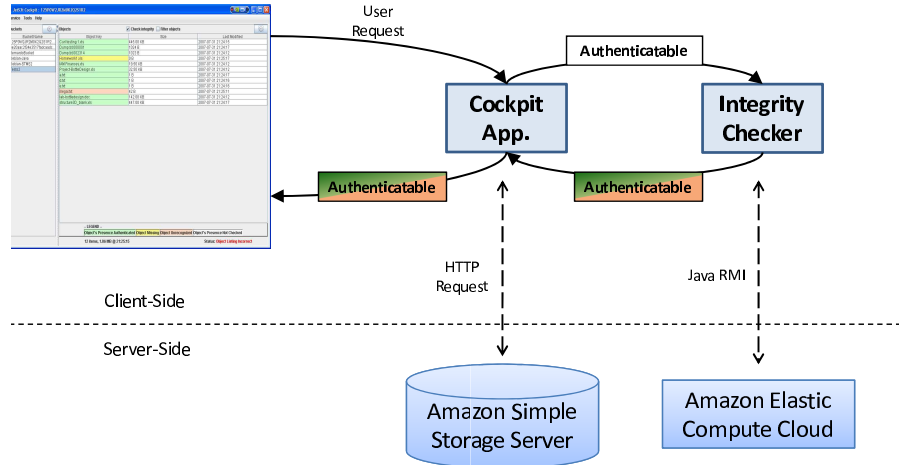


Figure 4.4: Software interaction architecture.

will provide identical functionality, though performance will undoubtedly differ. In fact, any integrity-checking service that can authenticate those four basic operations can easily plug-in to our prototype simply by extending `IntegrityChecker`. Information is passed between the GUI application and the integrity checker through objects implementing the `Authenticatable` (see Figure 4.5) interface. Implementing classes must be able to store and provide information about the authentication state of their objects’ contents, as well as their objects’ presence in a list.

While the authentication times for the four main operations are not insignificant compared to the time to complete the unauthenticated versions of these operations, the practical authentication time overhead depends to a large extent on the level of parallelism utilized in the implementation. To this effect, the authentication algorithms used in this prototype allow the network queries and computational operations for authentication to be conducted at the same time that data is being retrieved from S3. The approach to parallelization differs for each of the four operations:

**The PUT and DELETE operations** An important distinction to make with respect to operations that update, rather than retrieve information, is that a

---

```

public abstract class IntegrityChecker
    /** gets the authenticated hash of the contents of the object with the given key.
     * return a String, the correct hash, or null, if the proof returned from EC2 is incorrect. */
    protected abstract String getAuthenticatedFileHash(String key);
    /** retrieves from EC2 the correct results of a list operation with the given prefix,
     * starting point priorLastKey, and ending point lastKey.
     * return the correct listing. */
    protected abstract String[] getAuthenticatedList(String prefix, String priorLastKey, String lastKey);
    /** checks the integrity of the elements adjacent to the object with the given key
     * and digest, updates EC2 to include that object's information, and stores the new basis. 10
     * return true if the the correctness of the new basis is assured, false otherwise. */
    protected abstract boolean performPutUpdate(String key, String fileDigest);
    /** checks the integrity of the elements adjacent to the object with the given key
     * updates EC2 to remove that object's information, and stores the new basis.
     * return true if the the correctness of the new basis is assured, false otherwise. */
    protected abstract boolean performDeleteUpdate(String key);
}

public interface Authenticatable extends Comparable<Authenticatable>
    /** return an integer which should indicate the authentication state of this object's content, 20
     * namely whether its integrity is intact, corrupted, or unchecked. */
    public int getContentAuthenticationStatus();
    /** sets the authentication state of this object's content. */
    public void setContentAuthenticationStatus(int status);
    /** return an integer which should indicate the authentication state of this object's presence
     * in a list. If the object is present in the list, this state should indicate whether or
     * not its presence is authorized, and if it is not present, should indicate whether or not it should be. */
    public int getPresenceAuthenticationStatus();
    /** sets the authentication state of this object's presence. */
    public void setPresenceAuthenticationStatus(int status); 30
    /** return the string that is the name of the file or object that will be/has been authenticated. */
    public String getKey();
    /** sets the string that is the name of the file or object that will be/has been authenticated. */
    public void setKey(String key);
}

```

---

Figure 4.5: IntegrityChecker abstract class and Authenticatable interface.

### 4.3. IMPLEMENTATION

71

positive authentication result does not guarantee that the state of the relevant files on the storage server is correct. Rather, the only guarantee is that the updated basis stored on the client corresponds to the authenticated data structure in the correct updated state. In other words, to actually authenticate the contents or presence of files on the storage server, the client must make either a GET or a LIST query, respectively. The function of the authenticated update operations is simply to be sure that the stored basis is correct. The update of the storage server and the update of the authentication server are entirely separate. This fact means that it is easy to conduct both updates in parallel, simply by sending the two network requests at the same time. No comparison of results is necessary in this case.

**The GET operation** There are two components of the authenticated GET operation that could potentially introduce a noticeable overhead. Our first concern is that the client requests a download of a very large file (1+ GB), in which case simply computing the hash of the file’s contents after the download is complete will take a considerable amount of time. To overcome this difficulty we do not wait for the entire file to be downloaded. The hash of the file is computed in pieces while the file is being downloaded, a process which effectively does not add any authentication overhead. Our second concern is that retrieving a proof for a GET operation from the authentication server may, again, take a significant amount of time. Therefore, rather than waiting for a file downloaded from S3 to be available, and subsequently computing its hash value and proving its correctness, we retrieve the correct hash of the given file from the integrity checker *while* the file is being downloaded from S3. When this approach is combined with the hashing scheme described above, the only work left to do after the download is complete (and theoretically the only operation contributing to the time overhead) is a simple comparison of the calculated hash and the one retrieved from the integrity checker.

**The LIST operation** For the LIST operation, rather than waiting for the results to be returned from the storage server and then authenticating them with the integrity checker, we request from the integrity checker the list that is guaranteed to be correct, and make a parallel request for the unauthenticated list. Once again, all that is left to do is compare the two lists and keep track of any discrepancies.

In this case, however, there is the additional difficulty that lists may be very large. If we attempt to download authenticated and unauthenticated

lists tens of thousands of elements long, we must wait a considerable amount of time before we can even begin the comparison. In the interest of giving the client more immediate feedback, we retrieve both the authenticated and unauthenticated lists in smaller blocks of 1,000 elements. This approach slightly increases the total time to perform large **LIST** operations, but gives more regular feedback, and eliminates the possibility that we run out of memory maintaining information on tens of thousands of files.

## 4.4 Experiments

In this section, we present preliminary experiments conducted with Amazon S3 and EC2. We show that the time overhead that is added due to the authentication service is negligible. We also demonstrate the scalability of our service.

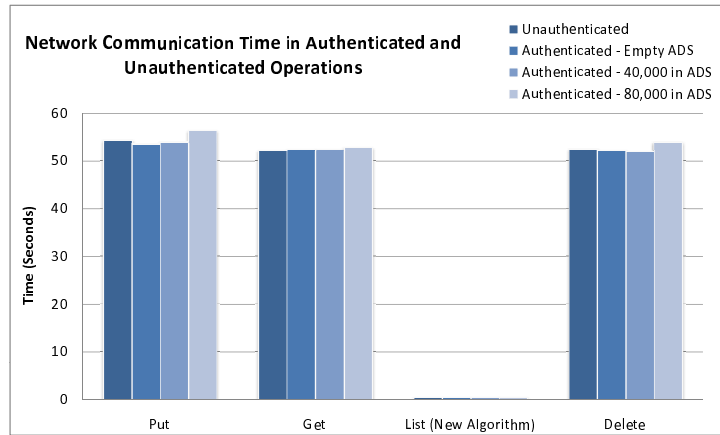


Figure 4.6: Comparison of non-authenticated and authenticated **GET**, **LIST**, **PUT**, and **DELETE** operations performed on a workload of 1,000 1K files, and with  $n = 0, 40,000$ , and  $80,000$ , averaging over 50 trials. Our new, efficient **LIST** implementation is used.

#### 4.4. EXPERIMENTS

73

##### Setup

We have implemented the authentication service in Java 1.5. Since we were not able to run the client on the same machine for all of the tests, two different machines were used. Machine 1 runs Linux, has 2G RAM, and an AMD Athlon X2 Dual Core 3800+ Processor. Machine 2 runs Windows XP, has 2G RAM and 2.16 GHz Intel Core Duo processor. The authentication server runs on a virtual machine (hosted by EC2) equivalent to a computer with a 1.0-1.2 GHz 2007 Opteron or 2007 Xeon processor, 1.7 GB of RAM, 160 GB of disk space, and 250 MB/sec of network bandwidth. The ping time from the client to the server on EC2 is roughly 13.72 ms for machine 1, and 40.25 ms for machine 2 (average of 10 trials). We denote with  $n$  the number of elements in the authentication and storage servers. We define the workload of the experiments to be the number of files whose content and/or authentication data is requested by the client, denoted with  $k$ , together with the size of the files when their content is requested. When reviewing these results, we must keep in mind that the vast majority of the run time is attributed to network communication, making them highly susceptible to variations in network speed.

##### Overhead Experiments

Figures 4.6 and 4.7 show the overhead added to the GET, LIST, PUT, and DELETE operations by our authentication service. We compare the completion times of the four unauthenticated operations with those of the four authenticated operations as we vary  $n$ . The workload is 1,000 1K files. Figure 4.6 displays the results of the test when run on machine 1. To demonstrate the efficiency of our LIST algorithm, we ran the same test on machine 2 using an older LIST implementation, the results of which are displayed in Figure 4.7. The procedure used to obtain the data in these figures was as follows: beginning with the original, unauthenticated version of Cockpit, a few lines of code were added to log the system time at the beginning and the end of each operation. A workload of 1,000 files of size 1K was uploaded to S3, a list of those elements was requested, the files were downloaded, and finally, the files were deleted. These PUT, LIST, GET and DELETE operations leave our S3 space in its original state, and we obtain the unauthenticated times for each operation. We repeat until we have the desired number of trials. Next, we run through the same procedure using our authenticated Cockpit, beginning with an empty authentication and storage servers. To show some degree of scalability, we repeat again with different values of  $n$ . The results are shown in Figures 4.6 and 4.7

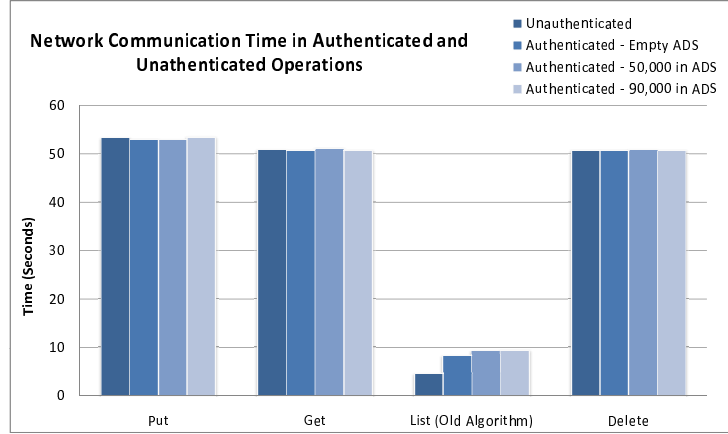


Figure 4.7: Comparison of non-authenticated and authenticated GET, LIST, PUT, and DELETE operations performed on a workload of 1,000 1K files, and with  $n = 0, 50,000$ , and  $90,000$ , averaging over 50 trials. An old, LIST implementation is used.

indicate that the time to execute the authenticated operations PUT, GET and DELETE differs by less than two seconds in each case from the time to execute the non-authenticated operations. Because of the uncertainty introduced by varying network conditions, it is difficult to say how much the authentication process contributes to the total operation time. As evidence, the authenticated time for many of the operations is actually smaller than the unauthenticated time, a result which can only be explained by variations in communication speeds. We can therefore say that within the precision range of our experiments, there is no time overhead for these operations. These results are a first indication that our service scales well (a topic that we will discuss further in Section 4.4). We would also like to highlight the improved efficiency of the LIST operation. The differences in the run conditions of the tests yielding the two graphs mean that they are not directly comparable. We can, however, compare the LIST times to the GET, PUT, and DELETE times in each individual figure. There are two main points of difference. Firstly, the new LIST completes drastically faster than the old compared to the other operations, even in the unauthenticated case. The primary cause of this change is that the new

#### 4.4. EXPERIMENTS

75

implementation has allowed us to increase the size of the list blocks from 100 to 1,000. Secondly, the older implementation of the LIST operation introduced significant authentication overhead, while our implementation appears to add no overhead at all. This result is not surprising, because as we discussed in Section 4.2, the computational and communication time for the new operation are both  $O(k + \log n)$ , a significant improvement over the  $O(k \log n)$  bound on the older operation.

#### Scalability Experiments

Figures 4.8, 4.9, 4.10, and 4.11, show how varying  $n$  affects the performance of our authentication service for the LIST, PUT, and DELETE operations. Figure 4.8 was obtained through tests on machine 1 with a workload of 1,000 1K files, varying  $n$  from 20,000 through 400,000 at increments of 20,000, while Figures 4.9, 4.10, and 4.11 are results of tests run machine 2, with a workload of 100 1K files, varying  $n$  from 10,000 through 200,000, at increments of 10,000. Figures 4.8 and 4.9 describe the scalability of the new and old LIST implementations respectively.

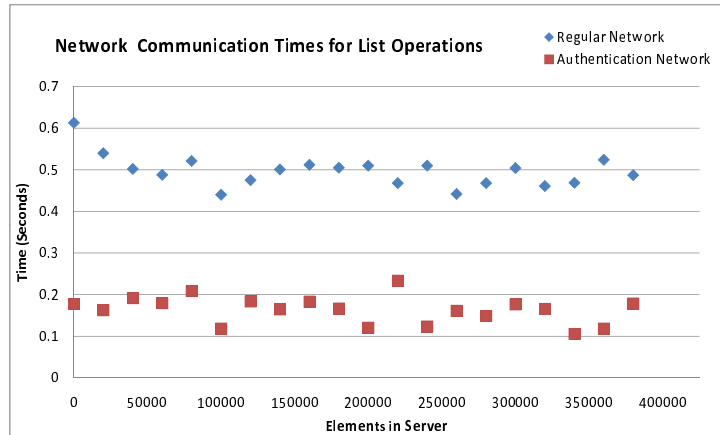


Figure 4.8: Authentication and regular network communication times for our new, efficiently authenticated LIST operation, varying  $n$  with a workload of 1,000 elements.

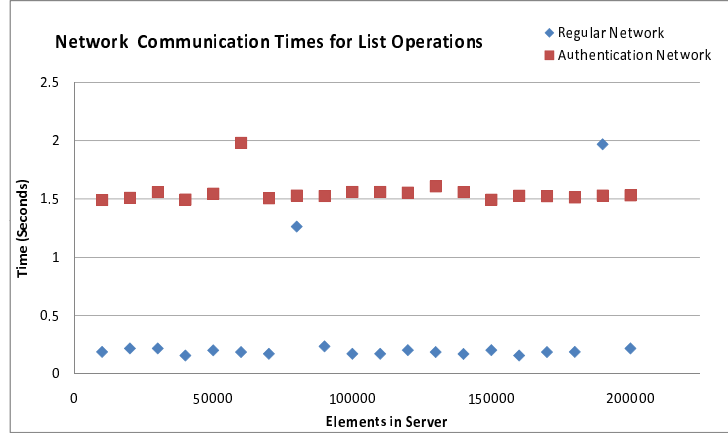


Figure 4.9: Authentication and regular network communication times for an old authenticated LIST operation, varying  $n$ , with a workload of 100 elements.

The procedure used to obtain these figures was as follows: We began with an empty authentication and storage servers. We wanted to time the operation varying  $n$  at intervals of  $i$ . For a workload  $k$ , we first uploaded  $k$  elements, then  $i - k$  elements. This step increases  $n$  by  $i$ . Next, we listed and then downloaded  $k$  elements. We repeated this operation for the desired range of  $n$ . We then deleted  $k$  elements, and then  $i - k$  elements, repeating until the authentication and storage servers are empty again. We separated each of the operations (LIST, PUT, and DELETE) into four parts: regular network (retrieval of the data from S3), authentication network (retrieval of the proof from EC2), query response (processing of query on EC2), and verification (processing of the proof on the client side). While the prototype is designed to maximize parallelism, performing the regular and authentication network queries concurrently, for these tests we separated the two components so that they run sequentially, allowing us to time them individually. During the operations, the Java garbage collector (GC) runs periodically. We have collected the GC run times and subtracted them from the times displayed in Figures 4.9, 4.10, and 4.11; Figure 4.8 is preliminary and does not take the GC into account.

We display only the regular and authentication network times for the LIST, PUT, and DELETE operations. We were unable to obtain reliable results for

#### 4.4. EXPERIMENTS

77

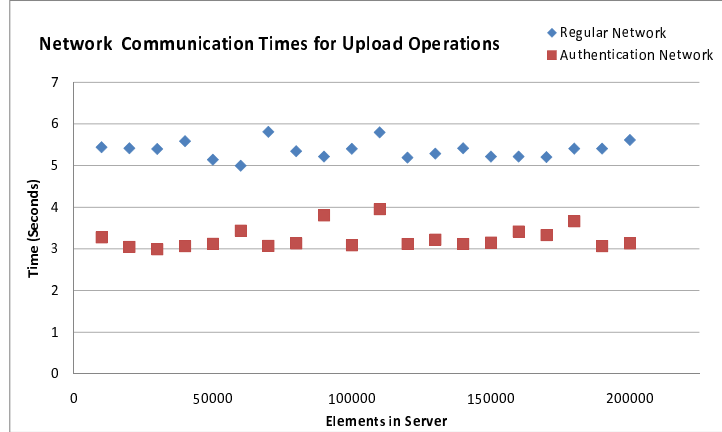


Figure 4.10: Times for the authentication and regular network components of an authenticated PUT operation, varying the number of elements. The workload in these experiments is 100 1K elements.

the GET operation because of the complicated interaction of threads during authentication, and the query and verification times account for only around one percent of the total time of each operation, making them somewhat irrelevant when considering the performance of the service. Ignoring the few outliers, and assuming that the odd peak in Figure 4.11 is caused by a spike in network traffic, one can see that the overall indication of these plots is that neither the regular or the authentication network operation time for LIST, PUT, or DELETE operations is affected significantly by the number of elements stored with our service. In other words, the service seems to scale extremely well. We can compare these plots to Figures 4.6 and 4.7 and see that the total times for each operation are very close to the larger of the authenticated and regular network operation times (the workloads vary, so we are actually comparing the time per element in  $k$ ). For the PUT and DELETE, and new LIST operations (Figures 4.8, 4.10, and 4.11), the regular network time is larger than the authentication network time, so we expect that when the regular and authentication network queries are performed in parallel, there will be no authentication overhead. In contrast, for the old LIST operation (Figure 4.9) the authentication network time is larger, so we expect some authentication overhead — once again the

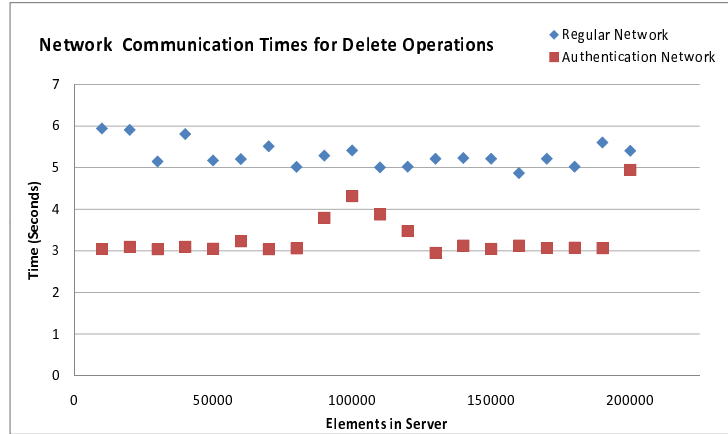


Figure 4.11: Times for the authentication and regular network components of an authenticated **DELETE** operation, varying the number of elements. The workload in these experiments is 100 elements.

improved efficiency of the new **LIST** implementation is evident.

## 4.5 Conclusions

This chapter presents the architecture and implementation of an integrity checking service that extends any existing online storage service. Our architecture is both space-efficient (the user stores only a single hash value) and time efficient (a very small overhead is added to the operations of the storage service). Our implementation is built on top of Amazon’s S3 and EC2 services. The experimental results confirm the negligible time overhead and scalability of our service.

## Chapter 5

# Graph Drawing for Security Visualization

### 5.1 Introduction

As an increasing number of software applications are web-based or web-connected, security and privacy have become critical issues for everyday computing. Computer systems are constantly being threatened by attackers that want to compromise the privacy of transactions (e.g., steal credit card numbers) and the integrity of data (e.g., return a corrupted file to a client). Therefore, computer security experts are continuously developing methods and associated protocols to defend against a growing number and variety of attacks. The development of security tools is an ongoing process that keeps on reacting to newly discovered vulnerabilities of existing software and newly deployed technologies.

Both the discovery of vulnerabilities and the development of security protocols can be greatly aided by visualization. For example, a graphical representation of a complex multi-party security protocol can give experts better intuition of its execution and security properties. In current practice, however, computer security analysts read through large logs produced by applications, operating systems, and network devices. The visual inspection of such logs is quite cumbersome and often unwieldy, even for experts. Motivated by the growing need for automated visualization methods and tools for computer security, the field of *security visualization* has recently emerged as an interdisciplinary community of researchers with its own annual meeting (VizSec).

In this chapter, we give a survey of security visualization systems that use

## 80 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

graph drawing methods. Thanks to their versatility, graph drawing techniques are one of main approaches employed in security visualization. Indeed, not only computer networks are naturally modeled as graphs, but also data organization (e.g., file systems) and vulnerability models (e.g., attack trees) can be effectively represented by graphs. In the rest of this paper, we specifically overview graph drawing approaches for the visualization of the following selected computer security concepts:

1. *Network Monitoring.* Monitoring network activity and identifying anomalous behavior, such as unusually high traffic to/from certain hosts, helps identifying several types of attacks, such as intrusion attempts, scans, worm outbreaks, and denial of service.
2. *Border Gateway Protocol (BGP).* BGP manages reachability between hosts in different autonomous systems, i.e., networks under the administrative control of different Internet Service Providers. Understanding the evolution of BGP routing patterns over time is very important to detect and correct disruptions in Internet traffic caused by router configuration errors or malicious attacks.
3. *Access Control.* Access to resources on a computer system or network is regulated by policies and enforced through authentication and authorization mechanisms. It is critical to protect systems not only from unauthorized access by outside attackers but also from accidental disclosure of private information to legitimate users. Access control systems and their associated protocols can be very complex to manage and understand. Thus, it is important to have tools for analyzing and specifying policies, identifying the possibility of unauthorized access, and updating permissions according to desired goals.
4. *Trust Negotiation.* Using a web service requires an initial setup phase where the client and server enter into a negotiation to determine the service parameters and cost by exchanging credentials and policies. Trust negotiation is a protocol that protects the privacy of the client and server by enabling the incremental disclosure of credentials and policies. Planning and executing an effective trust negotiation strategy can be greatly aided by tools that explore alternative scenarios and show the consequences of possible moves.
5. *Attack Graphs.* A typical strategy employed by an attacker to compromise a system is to follow a path in a directed graph that models vul-

## 5.2. NETWORK MONITORING

81

Table 5.1: Graph drawing methods used in the security visualization systems surveyed in this chapter.

	Force-Directed	Layered	Bipartite	Circular	3D
<b>Network Monitoring</b>	[32, 60, 70, 103]		[6, 18, 106]	[102]	
<b>BGP</b>	[101]			[101]	[79]
<b>Access Control</b>		[69]			
<b>Trust Negotiation</b>		[105]			
<b>Attack Graphs</b>		[77, 78]			

nerabilities and their dependencies. After an initial successful attack to a part of a system, an attacker can exploit one vulnerability after the other and reach the desired goal. Tools for building and analyzing attack graphs help computer security analysts identify and fix vulnerabilities.

In Table 5.1, we show the graph drawing methods used by the systems surveyed in this paper.

## 5.2 Network Monitoring

**Supporting Intrusion Detection by Graph Clustering and Graph Drawing [103].** In this paper, the authors use a combination of force-directed drawing, graph clustering, and regression-based learning in a system for intrusion detection (see Figure 5.1). The system consists of modules for the following functions: packet collection, graph construction and clustering, graph layout, regression-based learning, and event generation.

The authors model the computer network with a graph where the nodes are computers and the edges are communication links with weight proportional to the network traffic on that link. The clustering of the graph is performed with a simple iterative method. Initially, every node forms its own cluster. Next, nodes join clusters that already have most of their neighbors. The spring embedder algorithm [26] is used to draw the clusters and nodes within the clusters. Since forces are proportional to the weights of the edges, if there is a lot of communication between two hosts, their nodes are placed close to each other. Also, in the graph of clusters, there is an edge between clusters  $A$  and  $B$  if there is at least one edge between some node of cluster  $A$  and some node of cluster  $B$ . The layout of the graph of clusters and of each cluster are computed using the classic force-directed spring embedder method.

Various features of the clustered graph (including statistics on the node degrees, number of clusters, and internal/external connectivity of clusters) are

## 82 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

used to describe the current state of network traffic and are summarized by a feature vector. Using test traffic samples and a regression-based learning strategy, the system learns how to map feature vectors to intrusion detection events. The security analyst is helped by the visualization of the clustered graph in assessing the severity of the intrusion detection events generated by the system.

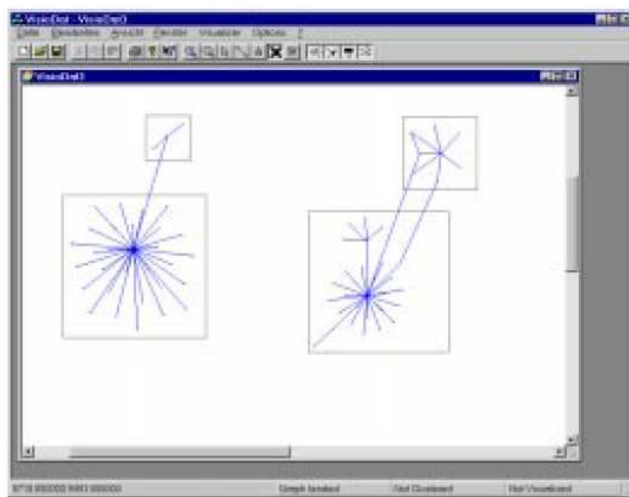


Figure 5.1: Force-directed clustered drawing for intrusion detection (thumbnail of image from [103])

**Visualization of Host Behavior for Network Security [60].** In this paper, the authors show how to visualize the evolution over time of the volume and type of network traffic using force-directed graph drawing techniques (see Figure 5.2). Since there are different types of traffic protocols (HTTP, FTP, SMTP, SSH, etc.) and multiple time periods, this multi-dimensional data set is modeled by a graph with two types of nodes: *dimension nodes* that represent traffic protocols and *observation nodes* that represent the state of a certain host in a given time interval. Edges are also of two types: *trace edges* that link

## 5.2. NETWORK MONITORING

83

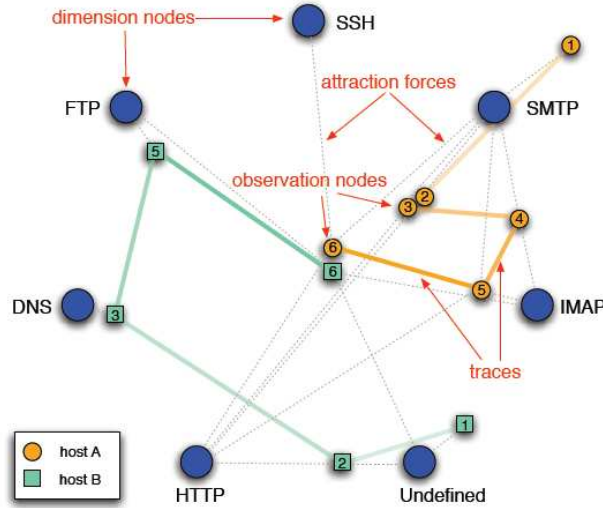


Figure 5.2: Evolution of network traffic over time (thumbnail of image from [60]): dimension nodes represent types of traffic and observation nodes represent the state of a host at a given time.

consecutive observation nodes and *attraction edges* that link observation nodes with dimension nodes and have weight proportional to the traffic of that type.

The layout is computed starting with a fixed placement of the dimension nodes and using a modified version of the Frucheterman-Reingold force-directed algorithm [29] that aims at achieving uniform edge lengths. The authors show how intrusion detection alerts can be associated with visual patterns in the layout.

**A Visual Approach for Monitoring Logs [32].** This paper (see Figure 5.3) presents a technique to visualize log entries obtained by monitoring network traffic. The log entries are basically vectors whose elements correspond to features of the network traffic, including origin IP, destination IP, and traffic volume. The authors build a weighted similarity graph for the log entries using a simple distance metric for two entries given by the sum of the differences of the respective elements. The force-directed drawing algorithm of [17] is used

## 84 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

to compute a drawing of the similarity graph of the entries.

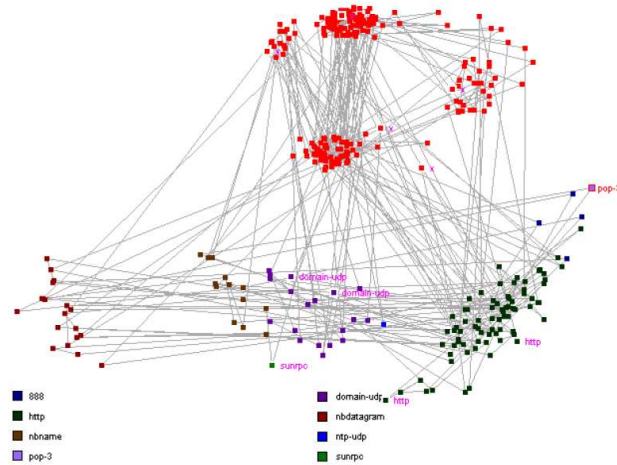


Figure 5.3: Similarity graph of log entries (thumbnail of image from [32])

### A Visualization Methodology for Characterization of Network Scans [70].

This work considers network scans, often used as the preliminary phase of an attack. The authors develop a visualization system that shows the relationships between different network scans (see Figure 5.4). The authors set up a graph where each node represents a scan and the connection between them is weighted according to some metric (similarity measure) that is defined for the two scans. Some of the features taken into consideration for the definition of the similarity measure are the origin IP, the destination IP and the time of the connection. To avoid displaying a complete graph, the authors define a minimum weight threshold below which edges are removed. The LinLog force directed layout method [76] is used for the visualization of this graph. In the drawing produced, sets of similar scans are grouped together, thus facilitating the identification of malicious scans.

**VisFlowConnect: NetFlow Visualizations of Link Relationships for Security Situational Awareness [106].** In this work, the authors apply

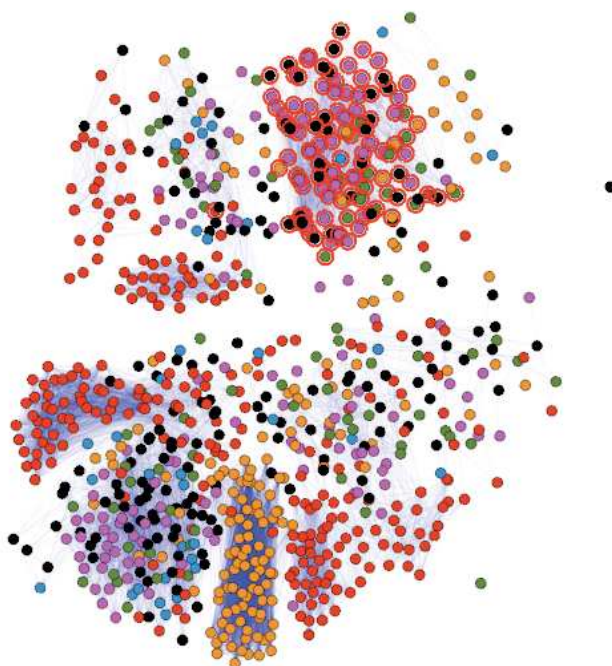


Figure 5.4: Similarity graph of network scans (thumbnail of image from [70]).

a simple bipartite drawing technique to provide a visualization solution for network monitoring and intrusion detection (see Figure 5.5). The nodes, representing internal hosts and external domains, are placed on three vertical lines. The external domains that send traffic to some internal host are placed on the left line. The domains of the internal hosts are placed on the middle line. The external domains that receive traffic from some internal host are placed on the right line. Each edge represents a network flow, which is a sequence of related packets transmitted from one host to another host (e.g., a TCP packet stream). Basically, the layout represents a tripartite graph. The vertical ordering of the domains along each line is computed by the drawing algorithm with the goal of minimizing crossings.

## 86 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

The tool uses a slider to display network flows at various time intervals and provides three views. In the global view, the entire tripartite graph is displayed to show all the communication between internal and external hosts. In the internal view and domain view, the tool isolates certain parts of the network, such as internal senders and internal receivers, and correspondingly displays a bipartite graph. The domain view and internal view are easier to analyze and provide more details on the network activity being visualized but on the other hand, the global view produces a high level overview of the network flows. The authors apply the tool in various security-related scenarios, such as virus outbreaks and denial-of-service attacks.

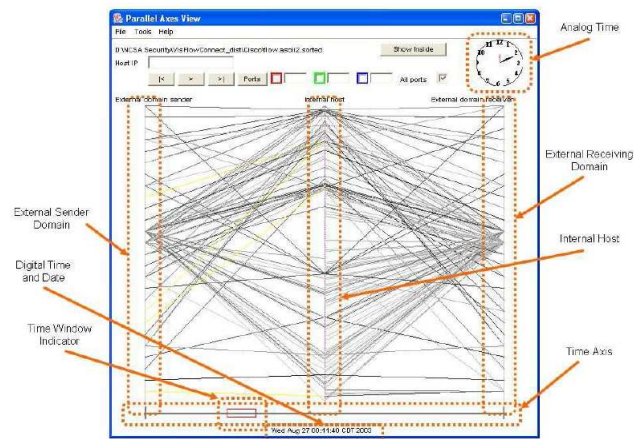


Figure 5.5: Global view of network flows using a tripartite graph layout: nodes represent external domains (on the left and right) and internal domains (in the middle) and edges represent network flows (packet streams) between domains (thumbnail of image from [106])

**Home-Centric Visualization of Network Traffic for Security Administration [6].** In this paper the authors use a matrix display combined with a simple graph drawing method in order to visualize the traffic between domains in network and external domains (see Figure 5.6). To visualize the internal net-

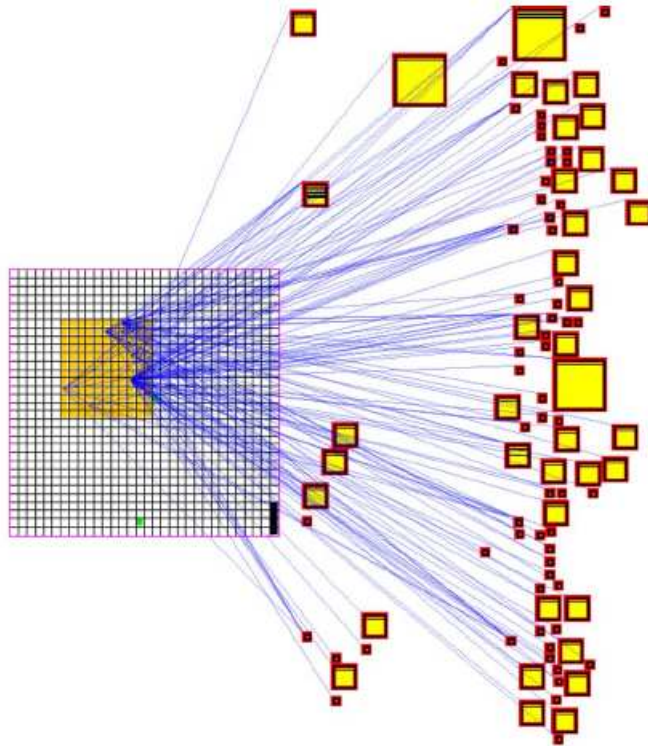


Figure 5.6: Visualization of internal vs external hosts using a matrix combined with a straight-line drawing. Internal hosts correspond to entries of the matrix while external hosts are drawn as squares placed around the matrix. The size of a the square for an external host is proportional to the amount of traffic from/to that host (thumbnail of image from [6]).

## 88 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

work, the authors use a square matrix: each entry of the matrix corresponds to a host of the internal network. External hosts are represented by squares placed outside the matrix with size proportional to the traffic sent or received. Straight-line edges represent traffic between internal and external hosts and can be colored to denote the predominant direction of the traffic (outgoing, incoming, or bidirectional). The placement of the squares arranges hosts of the same class A, B or C network along the same vertical line and attempts to reduce the number of edge crossings. Further details on the type of traffic can be also displayed in this tool. For example, vertical lines inside each square are used to indicate ports with active traffic. This system can be used to visually identify traffic patterns associated with common attacks, such as virus outbreaks and network scans.

**EtherApe: A Live Graphical Network Monitor [102].** This tool shows traffic captured on the network interface (in a dynamic fashion) or optionally reads log files like PCAP (Figure 5.7). A simple circular layout places the hosts in a circular shape and highlights network traffic between hosts by edges between them. Each protocol is distinguished by a different color and the width of the edges show the amount of traffic. This tool allows to quickly understand the role of a host in the network and the changes in traffic patterns during time. Beyond the graphical representation it is also possible to show detailed traffic statistics of active ports.

**Rumint: A Graphical Network Monitor [18].** Rumint (Figure 5.8) is a free tool available at <http://www.rumint.org/>. It takes captured traffic as input and visualizes it in various unconventional ways. The most interesting visualization related to graph drawing is the parallel plot that allows one to see at one glance how multiple packet fields are related. An animation feature allows to analyze various trends over time.

### 5.3 Border Gateway Protocol

**BGP eye: A New Visualization Tool for Real-Time Detection and Analysis of BGP Anomalies [101].** In this paper, the authors propose to use a new visualization tool, called *BGP Eye*, that provides a real-time status of BGP activity with easy-to-read layouts. BGP eye is a tool for root-cause analysis of BGP anomalies. Its main objective is to track the healthiness of BGP activity, raise an alert when an anomaly is detected, and indicate its

### 5.3. BORDER GATEWAY PROTOCOL

89

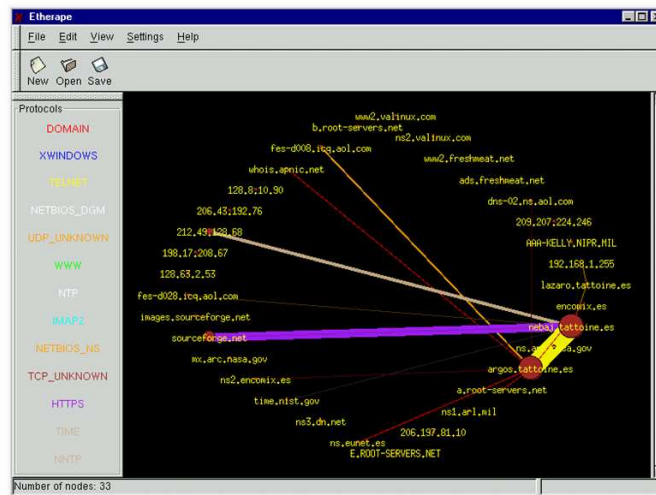


Figure 5.7: Traffic monitoring with Etherape (thumbnail of image from [102]).

most probable cause. BGP Eye allows two different types of BGP dynamics visualization: *internet-centric view* see Figure 5.9 and *home-centric view*, see Figure 5.10. The internet-centric view studies the activity among ASes (autonomous systems) in terms of BGP events exchanged. The home-centric view has been designed to understand the BGP behavior from the perspective of a specific AS. The inner ring contains the routers of the customer-AS and the outer ring contains their peer routers, belonging to other ASes. In the outer layer, the layout method groups routers belonging to the same AS together and uses a placement algorithm for the nodes to reduce the distance between connected nodes.

**VAST: Visualizing Autonomous System Topology [79].** This tool (Figure 5.11) uses 3D straight-line drawings to display the BGP interconnection topology of ASes with the goal of allowing security researchers to extract quickly relevant information from raw routing datasets. VAST employs a quad-tree to show per-AS information and an octo-tree to represent relationship between multiple ASes. Routing anomalies and sensitive points can be quickly detected, including route leakage events, critical Internet infrastruc-

## 90 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

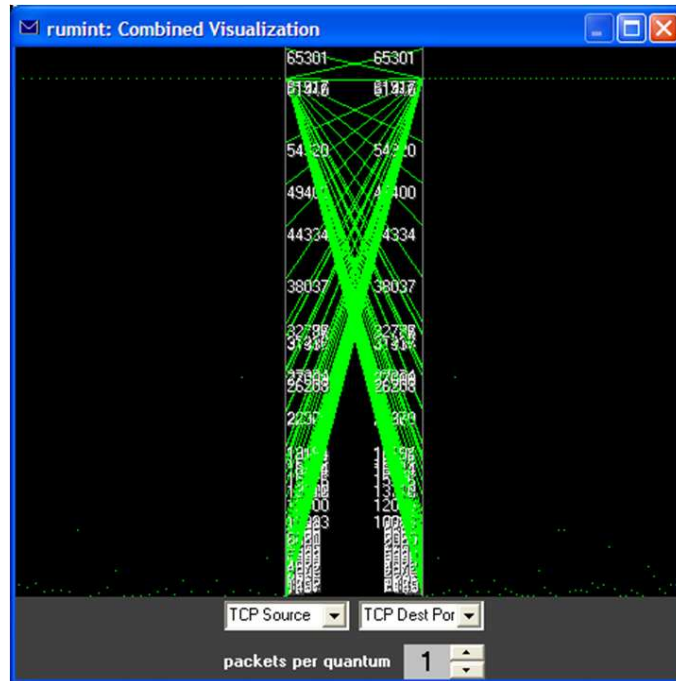


Figure 5.8: Visualization of an NMAP scan with Rumint (thumbnail of image from [18]).

ture and space hijacking incidents. The authors also developed another tool, called *Flamingo*, that uses the same graphical engine as VAST but is used for real-time visualization of network traffic.

**BGPlay and iBGPlay [9].** *BGPlay* and *iBGPlay* (Figure 5.12) provide animated graphs of the BGP routing announcements for a certain IP prefix within a specified time interval. Both visualization tools are targeted to Internet service providers. Nodes represent an AS and paths are used to indicate the sequence of ASes needed to be traversed to reach a given destination. *BGPlay* shows paths traversed by IP packets from a several probes spread over the Internet to the chosen destination (prefix). *iBGPlay* shows data privately col-

### 5.3. BORDER GATEWAY PROTOCOL

91

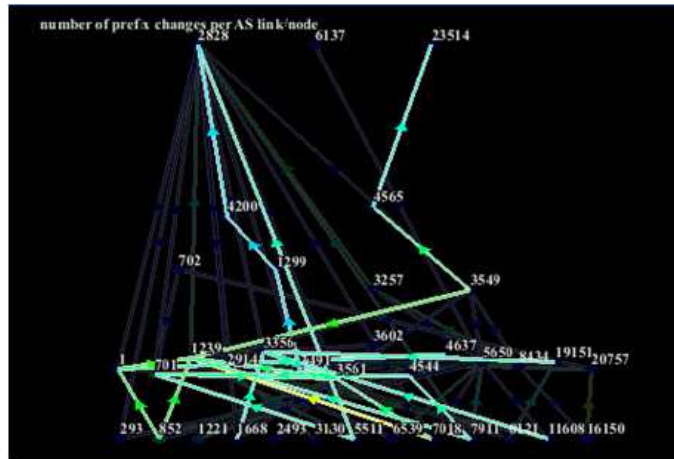


Figure 5.9: Internet-centric view view in BGP Eye (thumbnails of images from [101]).

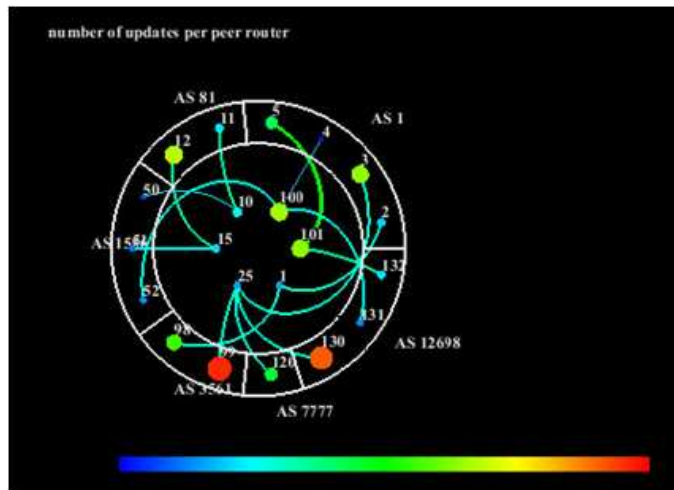


Figure 5.10: Home-centric view in BGP Eye (thumbnails of images from [101]).

## 92 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

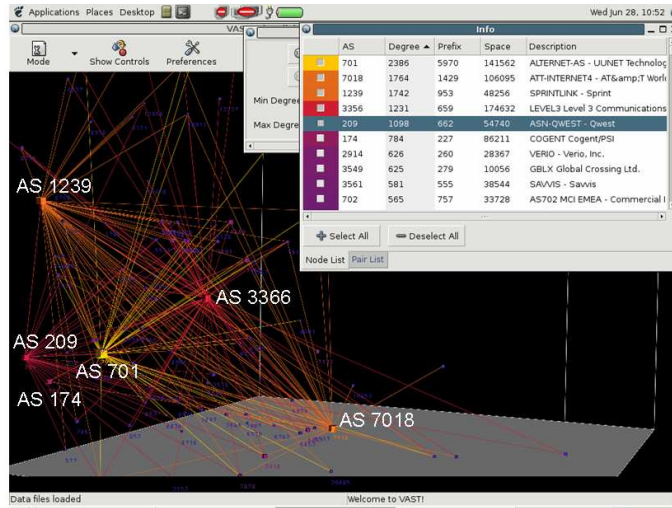


Figure 5.11: Largest autonomous systems in the internet visualized with VAST (thumbnail of image from [79]).

lected by one ISP. The ISP can obtain from iBGPlay visualizations of outgoing paths from itself to any destination. The drawing algorithm is a modification of the force-directed approach that aims optimizing the layout of the paths.

### 5.4 Access Control

#### Information Visualization for Rule-based Resource Access Control [69].

In this paper, the authors provide a visualization solution for managing and querying rule-based access control systems. They develop a tool, called RubaViz, which makes it easy to answer questions like “What group has access to which files during what time duration?”. RubaViz constructs a graphs whose nodes are subjects (people or processes), groups, resources, and rules. Directed edges go from subjects/groups to rules and from rules to resources to display allowed accesses. The layout is straight-line and upward.

#### 5.4. ACCESS CONTROL

93

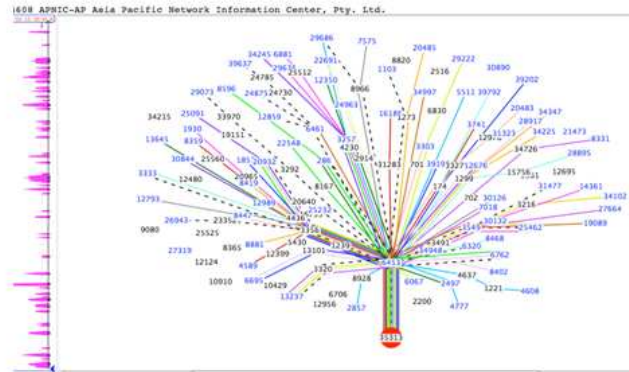


Figure 5.12: In BGPlay, nodes represent autonomous systems and paths are sequences of autonomous systems to be traversed to reach the destination (thumbnail of image from [9]).

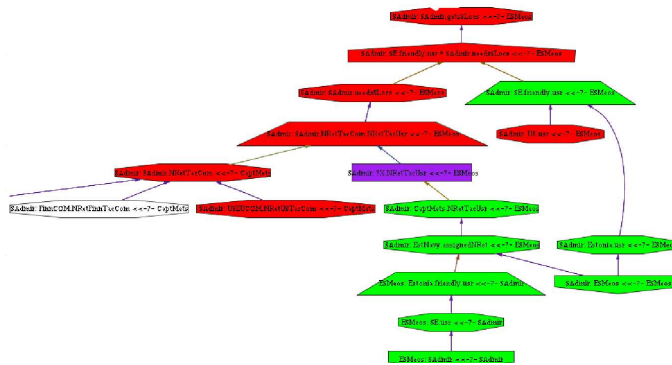


Figure 5.13: Drawing of the trust-target graph generated by a trust negotiation session (thumbnail of image from [105]).

## 94 CHAPTER 5. GRAPH DRAWING FOR SECURITY VISUALIZATION

### 5.5 Trust Negotiation

**Visualization of Automated Trust Negotiation [105].** In this paper, the authors use a layered upward drawing to visualize automated trust negotiation (ATN) (Figure 5.13). In a typical ATN session, the client and server engage in a protocol that results in the collaborative and incremental construction of a directed acyclic graph, called trust-target graph, that represents credentials (e.g., a proof that a party has a certain role in an organization) and policies indicating that the disclosure of a credential by one party is subject to the prior disclosure of a set of credentials by the other party. A tool based the Grappa system [7], a Java port of Graphviz [27], is used to construct successive drawings of the trust-target graph being constructed in an ATN sessions.

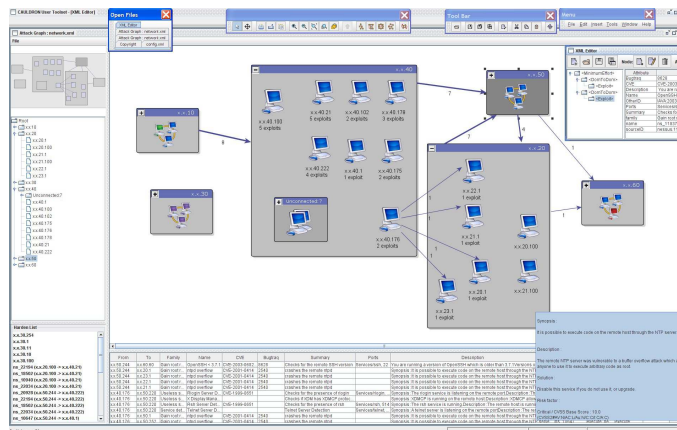


Figure 5.14: Visualization of an attack graph (thumbnail of image from [77]).

### 5.6 Attack Graphs

**Multiple Coordinated Views for Network Attack Graphs [77]** This paper describes a tool for visualizing attack graphs (Figure 5.14). Given a network and a database of known vulnerabilities that apply to certain machines of the network, one can construct a directed graph where each node is a machine (or group of machines) and an edge denotes how a successful attack on the

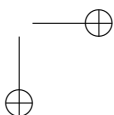
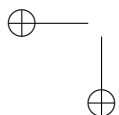
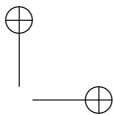
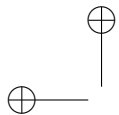
## 5.7. CONCLUSIONS

95

source machine allows to exploit a vulnerability on the destination machine. Since attack graphs can be rather large and complex, it is essential to use automated tools to analyze them. The tool presented in this paper clusters machines in order to reduce the complexity of the attack graph (e.g., machines that belong to the same subnet may be susceptible to the same attack). The Graphviz tool [27] is used to produce a layered drawing of the clustered attack graph. Similar layered drawings for attack graphs are proposed in [78].

## 5.7 Conclusions

In this chapter, we have presented a survey of security visualization methods that use graph drawing techniques. The growing field of security and privacy offers many opportunities to graph drawing researchers to develop new drawing methods and tools. In computer and network security applications, the input to the visualization system is often a large multidimensional and temporal data set. Moreover, the layout needs to support color, labels, variable node shape/size and edge thickness. In most of the security visualization papers we have reviewed, either simple layout algorithms have been implemented (e.g., spring embedders) or open-source software has been used (e.g., Graphviz). In order to make a larger collection of sophisticated graph drawing techniques available to computer security researchers, it is important for the graph drawing community to develop and distribute reliable software implementations.



## Chapter 6

# Effective Visualization of File System Access-Control

### 6.1 Introduction

Access-control configuration for computer systems is a critical task. Users expect to be given tools for protecting sensitive data stored in files and folders from unauthorized access. For example, a user  $A$  should not be able to perform an operation (e.g., read, write) on the files of a user  $B$  unless appropriate authorization was given by user  $B$ . Languages for expressing access control rules and policies and mechanisms for determining and enforcing the permissions a user has to access a given resource are fundamental problems that have been extensively studied.

The access control model employed by current-generation file systems, such as Microsoft Windows XP and Vista, is rather complex and often insufficiently documented. In a large file system with multiple users, it is rather tricky to understand which users/groups can access which files and with which permissions. Also, the effect of simple operations (such as copy and move) on the permissions of a file are difficult to anticipate and sometimes unintuitive. For example, consider a Windows user who changes the permissions of a certain file to make it not readable by others and later moves this file to another folder where the read permission is inherited from the parent folder. The user is unlikely to realize that after the move, the file is no longer protected. This is due to the fact that in a Windows NTFS file system, there are three types of permissions associated with a file: the *local* permissions for the file, the *inherited*

permissions derived from the permissions of the parent folder, and the *effective* permissions, obtained as the union of the local permissions and the inherited permissions.

Inherited permissions have many advantages and have been adopted by several file systems. For example, suppose all files and folders are set to inherit from their parent folder. If one wants to make readable a folder  $f$  and all its descendant files and folders, it is sufficient to apply the read permission only to  $f$ . The read permission is then recursively inherited by its descendants. However, inheritance can be tricky when copying and moving files, as a file can inherit new permissions that were not intended for the file. Hence, it is important to clearly show users which folders inherit from their parent folder and where inheritance is interrupted instead.

Also, as pointed out by Montemayor et al. [69], inherited permissions and other features of access control mechanisms can make answering questions such as “What group has access to which files during what time duration?” or “If I implement this policy, what conflicts this result?” very difficult.

Understanding file permissions and setting them to achieve desired file sharing and protection goals can be a daunting task for non-expert users and is non-trivial even for experts. A tool that helps users to understand how access-control permissions are determined and the effect of file system operations on file permissions would be extremely useful for both regular users and administrators.

We believe that an effective way to overcome difficulties of understanding file permissions is through visualization. Therefore, in this chapter, we present our preliminary design of a visualization tool that displays access-control information in a way that is easily understandable and helps the user set the correct permissions to achieve file sharing and protection goals. Our visualization tool uses treemaps [47], a popular graphical representation of hierarchical structures based on a recursive decomposition of rectangles into sub-rectangles.

The main related work is as follows. In Windows, advanced file system permissions are displayed as a list. Reeder et al. [89] propose using a square matrix to visualize the permissions of a file system and presents an example with changes of groups and users permissions. Montemayor et al. [69] present a solution for access control visualization based on representing the connections between groups, users, and resources, with a graph. The complexity of access control safety and the administrator’s difficulty in dealing with it (which makes visualization of access control very important) is analyzed in [45]. The usability of access control systems is discussed in [15]. Some visualization solutions for access-control and file-sharing policies are presented in [88].

## 6.2. PRELIMINARIES

99

Treemaps were introduced in 1991 [47] as a method of representing a complex hierarchy in a compact space. Bladh et. al [10] provide a file system visualization based on treemaps in the 3D space. Interactive ways to explore a file system through visualization are presented in [28]. Stasko [96] gives an evaluation of different compact ways to represent hierarchical structures. The visualization of dynamic hierarchies is presented in [104]. Finally, in 1971, a method using a nested rectangle representation (that resembles treemaps but though not formally defined) to visualize program execution was presented [48].

## 6.2 Preliminaries

In this section we introduce some preliminaries about the security aspect of our work (access control) and the visualization aspect of our work (treemaps).

### NTFS Access Control Management

In this part, we focus on the *access control list* (ACL) implemented in the Windows NTFS (New Technology File System). NTFS [90] allows to define access control information for each file system object. Using different security policies it is possible to allow or deny access to files and folders for determined users or groups. The file system driver manages all file system requests (i.e., create new files, open existing files, write to files. etc.) as the intermediary between the operating system and the storage device drivers.

The NTFS driver manages access to the file system according to defined permissions that are expressed by the ACL.

NTFS ACLs are composed of *access control entries* (ACEs). Each ACE allows or denies specific permissions (i.e., by a user or a group) to or from an object.

Starting with Windows 2000, NTFS allows to dynamically manage permission inheritance. That is, when you create a subfolder or a file in a NTFS folder, the child object not only inherits the parent’s permissions but maintains a kind of link with its parent. Furthermore, parent’s permissions are stored separately from any local permissions that are directly stored on the child. So for any changes performed on the parent folder, this method allows the child objects to automatically inherit the changes from their parents and to prevent from overwriting all the local permissions.

This approach allows an administrator or a user to manage a hierarchical tree of permissions that matches the directory tree. Since each child inherits

permissions *recursively* from its parent. So it is possible to perform changes of permissions with little effort.

The main downside of *dynamic* inheritance is the increase of complexity and the possibility to have conflicting ACEs. The NTFS security module combines the specified permissions (i.e. local and inherited ACEs, allows or explicit denies) and decides whether to grant or deny the access to a user, a group, or other security entities. Microsoft introduced in Windows XP the *effective permissions* tab to help the administrator in the quite tricky task of understanding the effective permission for a user or a group on a specific file system object.

### Treemaps

Treemaps (see, e.g., [11, 47]) were introduced in 1991 as a way to represent large hierarchical structures in a compact way. The main idea of creating a treemap can recursively be described as follows: Given a tree  $T$  with root  $r$ , assign a rectangle  $A$  to represent  $T$ . Then, for all the subtrees  $T_1, T_2, \dots, T_k$  of  $r$ , partition  $A$  into  $k$  rectangles  $A_1, A_2, \dots, A_k$  and assign  $A_1, A_2, \dots, A_k$  to  $T_1, T_2, \dots, T_k$ . This process continues until it reaches the leaves, where it assigns distinct rectangles for every leaf of the tree. Given a tree with  $n$  nodes, a treemap can be constructed in  $O(n)$  time using a bottom-up traversal.

Several algorithms have been proposed for assigning rectangles to subtrees. The standard method is based on the “slice-and-dice” algorithm, originally introduced in [47], which uses parallel lines to divide the rectangle assigned to a subtree  $T$  into smaller areas that correspond to the subtrees of  $T$ . It also alternates the direction of the parallel lines (horizontal/vertical) from one level the next, so that the change of levels is displayed. The standard treemap method often gives thin, elongated rectangles. A new method—the “squarified” algorithm—is presented in [11] to generate layouts in which the rectangles approximate squares.

## 6.3 Effective Access Control Visualization

In this section, we present the main features of the tool we have designed to assist administrators and users in better understanding and managing access-control of a hierarchical file system. The tool employs treemaps to visualize the file system tree. We use colors to distinguish the permissions of files and folders, and we indicate where a *break* of inheritance occurs with a special border around the relevant node in the treemap. The input to our tool consists of two items:

### 6.3. EFFECTIVE ACCESS CONTROL VISUALIZATION

101

1. The “user” input, which indicates the user or group whose permissions we are interested in. In Figure 6.1, this is indicated with the label “name”.
2. The “baseline” input, which basically indicates a certain combination of permissions upon which the color scheme of our visualization is based. In Figure 6.1, this is indicated with the label “permission”.

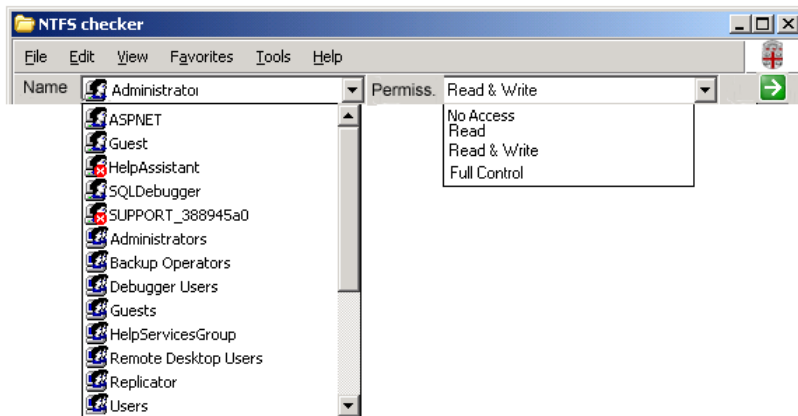


Figure 6.1: The user interface of our visualization tool. The main screen consists of the “user” input and the “baseline” input.

In the current design, the “baseline” input can take four values, namely the values **no access**, **read**, **read&write**, **full control**. These are sorted in “increasing permission” order. The user can also propose (and insert into the drop-down menu) another combination of permissions (e.g., **read&execute**) and the administrator is responsible for putting the new feature in the correct order (see Figure 6.1). The visualization tool reads this value and parses the file system tree, building the treemap using the slice-and-dice algorithm. For every file encountered, the associated node in the treemap is painted green, red, or gray, if the file’s permissions are weaker (more restrictive), stronger (less restrictive), or the same as those specified by the baseline, respectively. The tool could potentially use different shades of the same color to declare intensity of permissions. Finally, the tool draws an orange border around treemap nodes associated with files or folders where inheritance is broken.

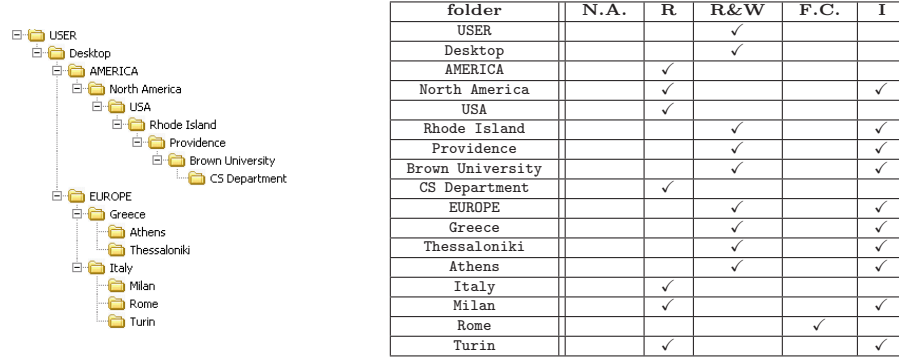


Figure 6.2: The directory tree that we are going to visualize with our tool, as visualized by Windows explorer. Beside the tree, we also show the effective permissions of each folder of the tree. **N.A.** stands for “no access“, **R** stands for “read“, **R&W** stands for “read and write“, **F.C.** stands for “full control” and **I** stands for “inheritance“.

We believe that this scheme makes it is easy to gain a general sense of current permissions of the file system as far a certain user is concerned. Furthermore, a more detailed understanding can be achieved simply by exploring the treemap more thoroughly. For example suppose that a file is moved from a folder that has weaker permissions than the baseline to a folder that has stronger permissions than the baseline. The administrator, by using our tool, will be able to notice that difference (since a small green area will appear in a greater red area). There is no longer a need to manually (by exploring the directory with `cd` commands) find files with changed permissions, a task that quickly becomes arduous as more users and other commands such as `copy` or `cacfs` are taken into consideration.

### Example

We show examples of using our tool to visualize the permissions of the directory tree of Figure 6.2. Figure 6.2 shows a directory tree and the effective permissions of every folder contained in this tree. We show in the table of Figure 6.2 four kinds of permissions, namely the permissions **no access**, **read**, **read&write**, **full control**. Also in the table of Figure 6.2 we have a column that indicates whether the certain folder inherits the permissions or not (the

#### 6.4. THE TRACE (TREEMAP ACCESS CONTROL EVALUATOR) TOOL.

103

last column).

In Figure 6.3(a) we see the representation of our file system with the treemap colored with colors according to permissions, as defined before. In Figure 6.3(b) we see the treemap layout of the file system after moving a file into a directory that has different permissions from the file. Also, in Figure 6.4(a) we see the treemap layout of a copy operation and in Figure 6.4(b) we see the treemap layout of the file system where the permissions of the root node of the directory have changed. Also note that in the presented visualizations we distinguish between the *local* and *effective* permissions. Namely, if the local and effective permissions coincide the tiles are painted with only one color. When this is not the case, we use the upper right corner to indicate the inherited permissions and the bottom left corner to indicate the local permissions. In this way we have a good overview of the permissions that correspond to a file. Note that the figures do not present the break of inheritance of a file since this will clutter up the space. The frames have been produced with the software from University of Maryland (<http://treemap.sourceforge.net/>), where we use the slice-and-dice algorithm for the layout and the increased border option to better display the directory structure.

#### 6.4 The TrACE (Treemap Access Control Evaluator) Tool.

The main goal of TrACE is to assist administrators and users in better understanding and managing access-control of a hierarchical file system. The tool employs treemaps to visualize the file system tree. A snapshot of TrACE is shown in Figure 6.5. TrACE indicates where a *break* of inheritance occurs with a special border around the relevant node in the treemap. We use colors to distinguish the permissions of files and folders. The background color of the rectangle associated with a file system object represents the object’s effective permissions, while a small colored square and circle in the top right corner of each rectangle indicate the contributions of the inherited and explicit permissions, respectively.

We provide two different coloring schemes. In the “full spectrum” scheme, each permission level is statically mapped to a single color. In the “baseline” scheme, a specific combination of permissions is dynamically chosen by the user to be the baseline, and the color assigned to other permission levels is determined by their strengths relative to that baseline. Finally, in order to highlight the most relevant areas of the file system, we adjust the size of each

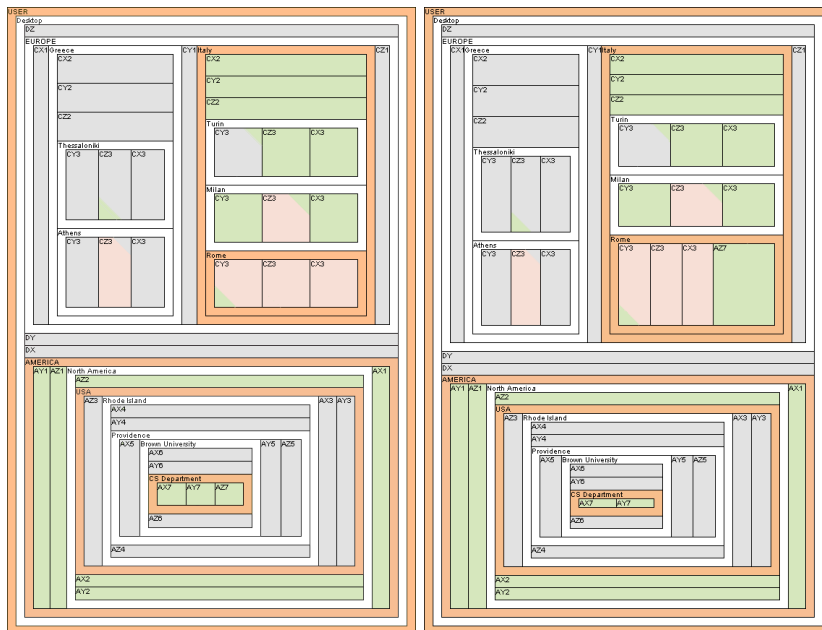


Figure 6.3: Treemap (a) shows the access control permissions for user *administrator*. User *administrator* has permission **R** on file **AZ7** because the color is light green, indicating a weaker permission compared with the baseline **R&W**. One can see that these permissions were inherited by *CS Department* because it is the first orange parent folder, indicating a break of inheritance. Treemap (b) illustrates the result of moving file **AZ7** from the *CS Department* folder to the *Rome* folder. The permissions of this file, indicated by the light green color, are preserved after the move. Furthermore, the size of the rectangle associated with the moved file increases to accentuate the move. Also, the color of the top right corner (the inherited permissions) of file **AZ7** is light green because after the move there is no inheritance from the parent.

#### 6.4. THE TRACE (TREEMAP ACCESS CONTROL EVALUATOR) TOOL.

105

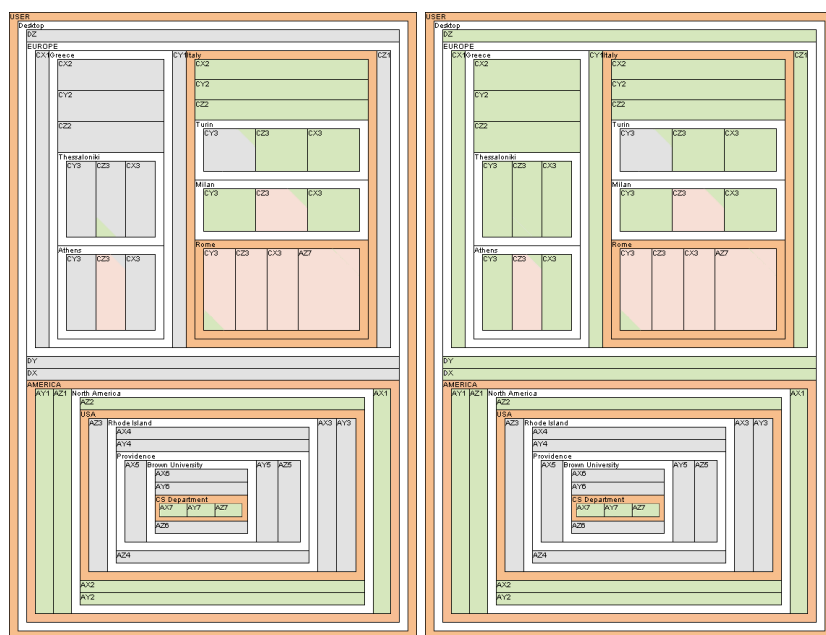


Figure 6.4: Treemap (a) shows the result of copying file **AZ7** from the *CS Department* folder to the *Rome* folder. The permissions of the file change, as indicated by the light red color, because of the inheritance from the destination folder. Treemap (b) shows the result of changing the permissions of the *USER* folder from **R&W** to **R**. This change propagates down to descendant files and folders until there is a break of inheritance. Note that file **CZ3** in the *Thessaloniki* folder changes its color from *grey* to *light green* because the local permission (left bottom corner) has a level that is lower than that of the inherited permission (top right corner). It is possible to see the opposite behavior in file **CZ3** in the *Athens* folder, where the inherited permission changes from *grey* to *light green* but the color of the rectangle remains *light red* because the level of the local permission is greater than that of the inherited permission.

file or folder in the treemap by counting the number of access control changes on the file or on the subtree of the folder. Thus, if the all the items in a folder have the same permissions, that folder will be very small in our visualization, but if many of the items in the folder have different permissions than their parents, then the folder will be large, allowing the user to detect the changes more easily. We can control the tool by specifying the following properties:

1. The user or group whose permissions we are interested in.
2. The folder whose sub-tree we would like to visualize.
3. The style—which coloring scheme to use. If the baseline coloring scheme is used, the user must also select the baseline.

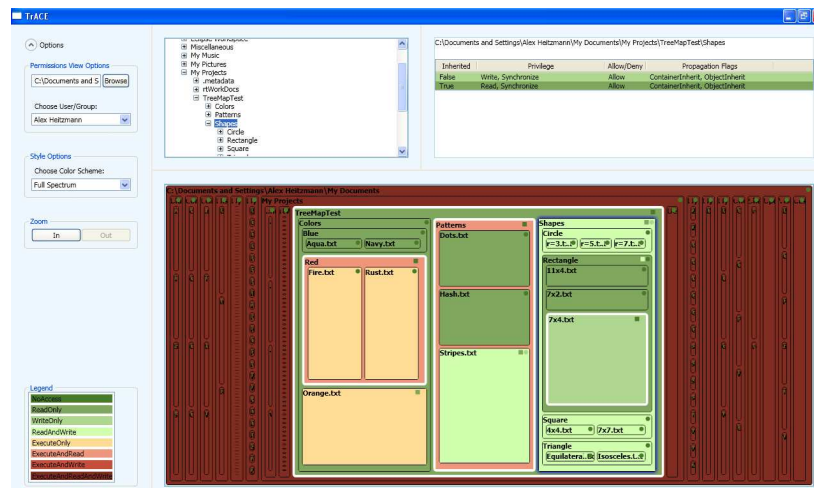


Figure 6.5: Snapshot of TrACE, illustrating how the tool emphasizes areas of interest within large file system trees.

## 6.5 User Feedback

In this section we present the results of a survey designed to test the effectiveness of TrACE in aiding users to identify important access control information

## 6.5. USER FEEDBACK

107

in Windows.

### Survey Details

Participants in the study included attendees of the vizSec/RAID 2008 and SMAU Milano 2008 conferences, as well as PhD students and faculties from the Roma Tre DIA network lab, undergraduates in the Brown University department of computer science CS 167/9 operating system class and graduates students in the Roma Tre University class of computer security. Participants were asked to fill out a questionnaire, a copy is present in the appendix, which included questions in three different categories:

- First category contained **background** questions designed to gauge levels of practical and theoretical familiarity in general with file permissions and in particular with NTFS access control.
- Questions in the second category required participants to use a screen snapshot, a copy of this is also available in the appendix) of a TrACE visualization of a directory tree to determine access control status of certain folders and files. This **usage** section of the survey focussed on analysis of permissions inheritance, as this is one of the main focuses of TrACE.
- Third category contained questions about User Interaction **design** choices, probable usage patterns, and general feedback.

Before being asked to fill out this survey, participants listened to presentations and demonstrations (10 - 20 minutes in conferences, 50 minutes in classes) about NTFS access control and TrACE. We provided clarification of general concepts in NTFS access control on request during the survey, but no help directly applicable to the problems in the second category was given, ensuring that the participants answers were their own.

### Results

The results for questionnaire answers are showed in *bar chart* graph in which each bar indicates a different conference. Each bar represents 100% of the amounts for that category. All the bars are stacked to analyze distributions within a single user study, and at the same time to compare the differences between more user studies in: conferences, demos, posters, lessons, and etc.

The results show only when there is an answer and they do not show when all the check boxes in a question were left empty.

### Background User Study

We used four questions to identify the background of the users and to understand which is the knowledge of how the file permissions work in an operating system. The first question (see Fig. 6.5) shows the different operating systems used by persons that answered to the questionnaire, it is interesting to see that Windows operating system is by far the most used in commercial environment like SMAU and graduate university class in Italy, while in research environment and in undergraduate American university Linux and MacOS are used more.

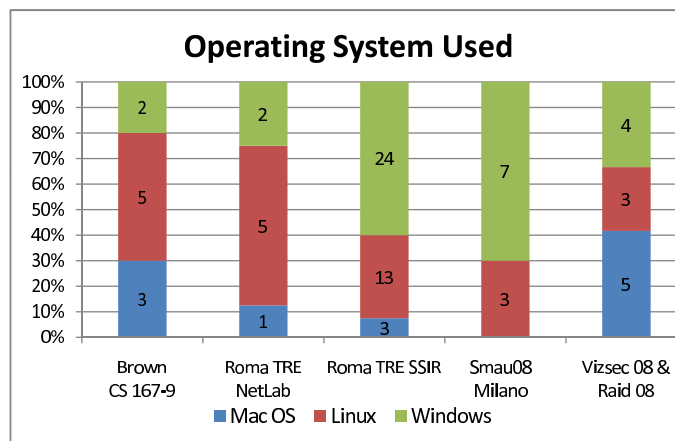


Figure 6.6: Question: Which operating system do you use primarily?

The second and third question (see Fig. 6.5 and Fig.6.5 ) show how often participants check and modify file permissions on their systems. As we expected users check permissions more often when they want to modify them and also a commercial or high research user is more careful about permission change. The fact that all the students in CS 167-9 course have tried to change permission at least once is probably because they were following an operating system course.

The last question (see Fig. 6.5) is the more interesting for the purpose of the user study because it asks directly the level of knowledge of NTFS to each

## 6.5. USER FEEDBACK

109

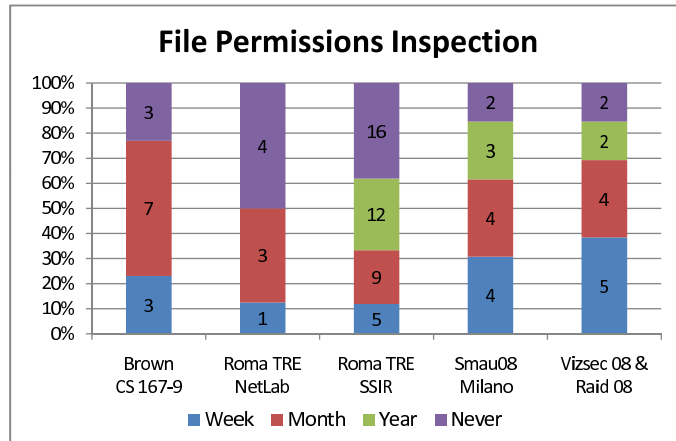


Figure 6.7: Question: How often do you inspect file permissions?

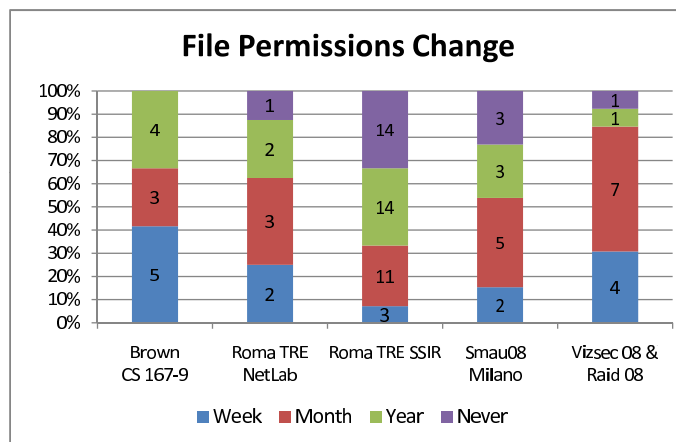


Figure 6.8: Question: How often do you change file permissions?

user. So this information it is very useful to rate if our tool is easier for a novice to NTFS or for an experienced user.

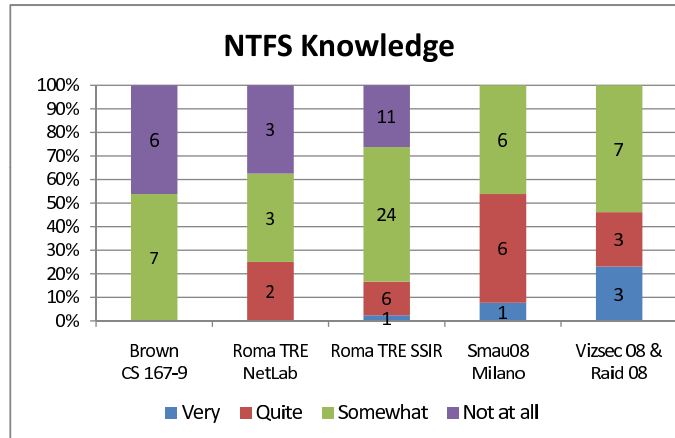


Figure 6.9: Question: How familiar are you with the NTFS access control schema?

### Usage User Study

We used four questions to understand how much TrACE is easy to use. These questions concern the inherit and explicit permissions rules in NTFS file systems. The results show the success percentage to practical questions in which we used a TrACE snapshot and we asked if they were able to understand some particular permissions on particular files or directories. It is important to say that the tool explanation usually did not take more than 20 minutes during class lesson and 10 minutes during conferences.

The first question (see Fig. 6.10) concerned specifically the problem of inheritance form a folder in particular a user should understand if permissions on a file were inherited or not. It is interesting to note that almost each group answered correctly with a rate higher than 50% this is very encouraging if less than 10% knew very well NTFS permissions.

The second question (see Fig. 6.11) concerned the problem of permission hierarchy. So we asked where is the origin of a particular permission. Essentially the participant should have understood the hierarchical algorithm to check effective permission. It is interesting to note that the percentage of success is higher than in previous question, probably because also the users that change permissions just sometimes are more sensitive to understand the origin of a permission that complex mechanisms of inheritance.

## 6.5. USER FEEDBACK

111

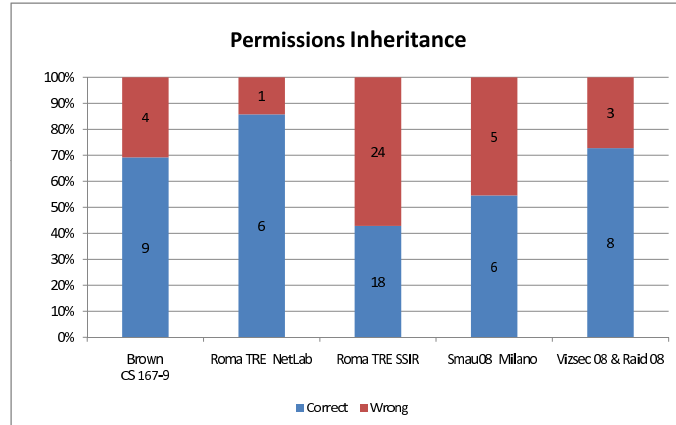


Figure 6.10: Question: Is a particular permission of a certain file in a defined folder inherited?.

The third question (see Fig. 6.12) concerned the problem of propagation of permission changes. So we asked what happens to all the files and subdirectory if we change a permission in the directory where they are contained. It is interesting to note that the percentage of success is higher in research level users but standard users have more difficult and the results are worst, probably because the propagation of changes is more a research problem than an everyday issue.

The fourth question (see Fig. 6.13) concerned the task of understanding the meaning of a difference of color between a file and its parent folder. The problem was about the explicit permissions. The trend of previous question is confirmed also if the VizSEC and RAID conference participants obtained the worst score.

What we understood from this section was that probably the TrACE tool is more suitable for a user with an academic background and this moves us to confront more with the industry to better understand what elements are more easily understandable, in this case, the low result obtained in SMAU conference is particularly relevant.

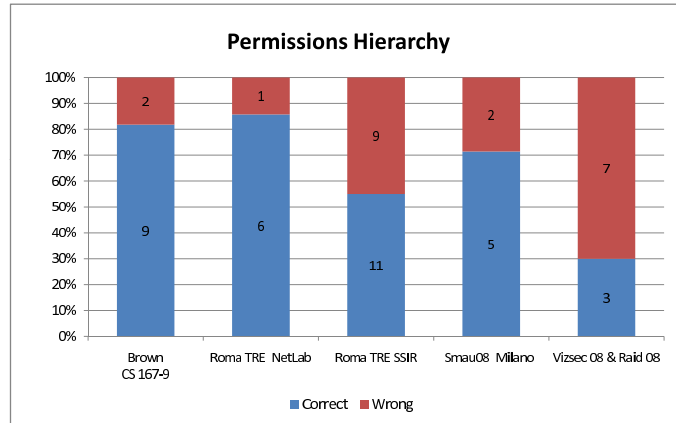


Figure 6.11: Question: Where did the permission for a particular file or folder originate?

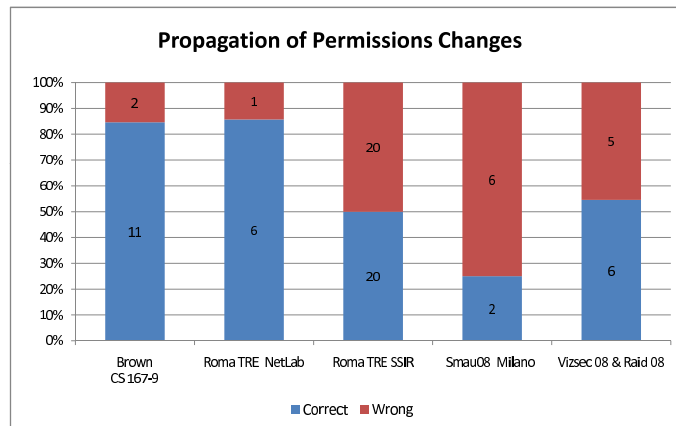


Figure 6.12: Question: If we change the permissions on a defined folder from a permission to another with more privileges, the permissions on a particular file will change?

## Design User Study

The last section of the user study asks to the user what do they think about the design choices in TrACE. The first two questions (see Fig. 6.14 and Fig.6.15)

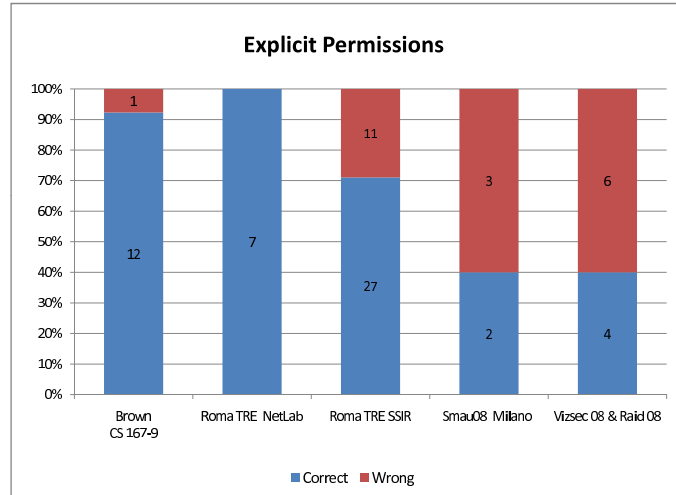


Figure 6.13: Question: Why does a particular file have a different color than its parent folder?

concern the color used to indicate more file permissions rights between the *green* and the *red* colors. Essentially, from the point of view of an administrator is more natural to figure out that *red* color is something dangerous so probably it has more privileges. At the opposite a user could imagine that *red* color shows something that is forbidden like traffic lighter. Unfortunately the user study did not help very much to understand this problem, because we did not have a one way decision. So essentially we obtained a slight advantage in the user point of view, but it is not enough and so we decided to put in the new version an option to switch colors. So any user can choose.

Next three questions (see Fig. 6.16, Fig. 6.17 and Fig.6.18) asked about what tool is easier and more preferred to use between Full Spectrum and Base-line visualization. So, what we understood is that Full Spectrum is easier for a novice and then we decided to use this visualization as starting screen. Last question concerns the usefulness of researching a new method to visualize file permissions in WIndows NTFS file system and also if it would be useful inside Window File Explorer. The result of this question is strongly encouraging to continue this research and even more because it is the only result that all the different communities which we proposed this questionnaire agree.

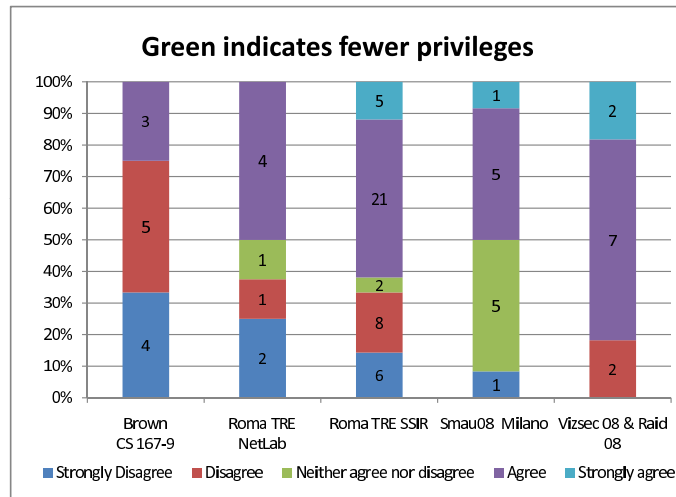


Figure 6.14: Question: The color green is a good choice to indicate fewer privileges, because it is safer for the administrator.

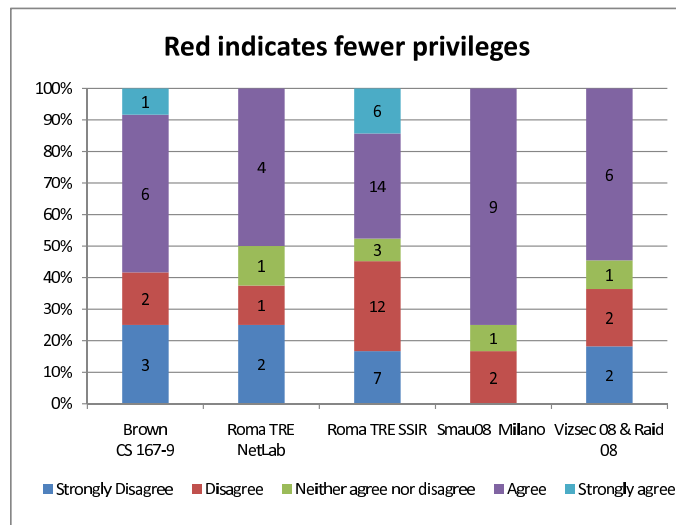


Figure 6.15: Question: The color red is a good choice to indicate fewer privileges, because it is more restrictive for the user.

## 6.5. USER FEEDBACK

115

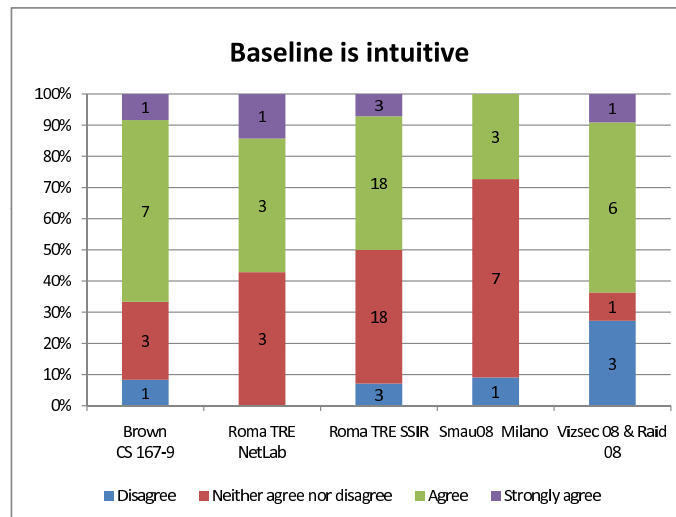


Figure 6.16: Question: The “baseline” display scheme is intuitive.

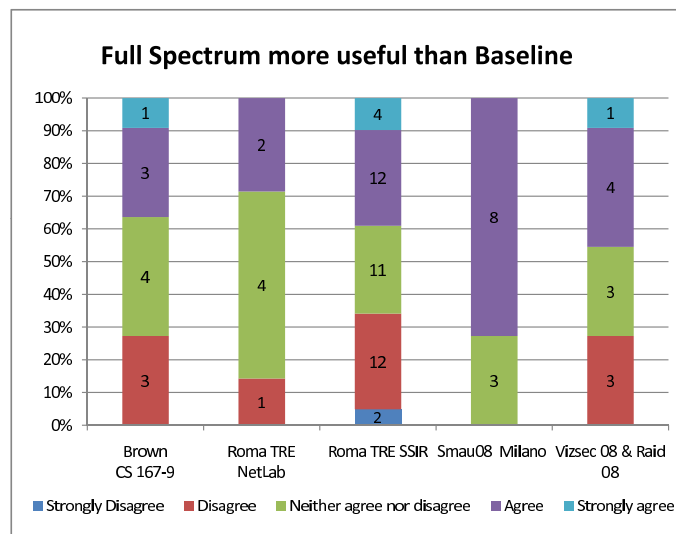


Figure 6.17: Question: The “full spectrum” display is more useful than “baseline” in locating incorrect access control configurations.

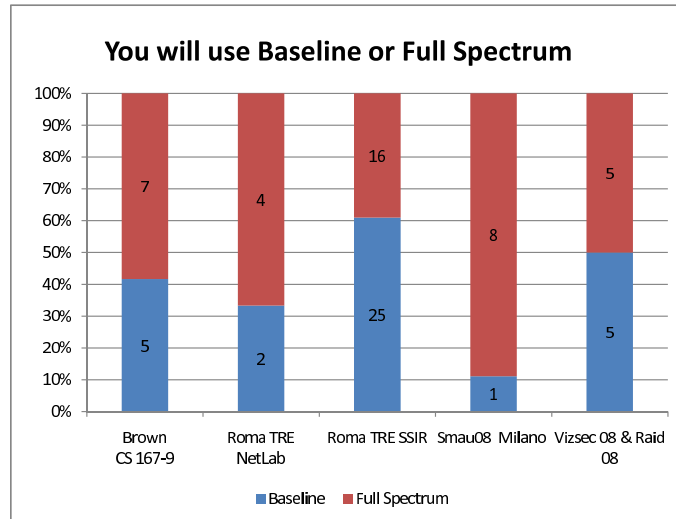


Figure 6.18: Question: Which of the two display schemes - ”full spectrum” or ”baseline” - would you be likely to use most often?

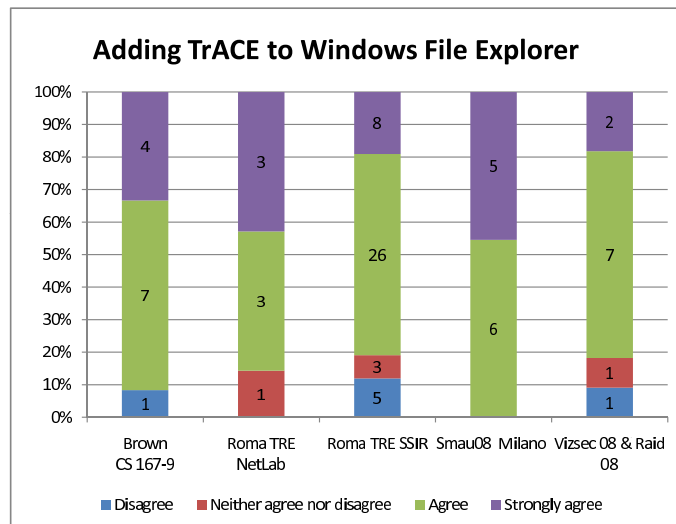


Figure 6.19: Question: If a visualization feature like TrACE was integrated into Windows File Explorer, a user could better understand and manage file system permissions.

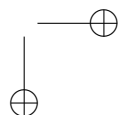
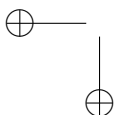
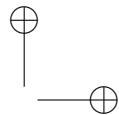
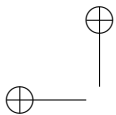
## 6.6. CONCLUSIONS

117

The questionnaire left some lines to leave some free comments or suggestions on TrACE tool, as usual, we received just a few of answers. Essentially the suggestions were focused on changing the number of colors used by using a rainbow instead of just *red* and *green* colors, further we received some suggestions to use a different treemap algorithm.

## 6.6 Conclusions

TrACE is a prototype tool for visualizing access control information for the Windows NTFS file system. Moving forward, we plan to develop further variations of the treemap layout which can display additional file permission information, to add functionality to *manage* permissions rather than simply analyzing them, and to support different file system types. The user study was very useful to understand that the tools that are available to inspect and change file permission in NTFS file system by default do not satisfy user needs completely. So, on the behalf of the user study we think that the use of visualization techniques could greatly help to build more usable tools for file permission administration.



# Conclusion

## Summary of Results

In this dissertation we presented an extensive study of data authentication and introduces a general method, based on a security *middleware*, external to the service, that performs authentication operations in parallel with standard service functions to minimize the time overhead. We examined the problem for different services, and designed efficient new techniques with authenticating general classes of operations, such as relational primitives, multidimensional queries and *relational join* and remote storage management. In particular, we addressed the problem of authenticating data in outsourced services, when a user stores more or less confidential information in a remote service such as an online calendar, remote storage, outsourced DBMS, and others.

We proposed an architecture and an implementation of an integrity checking service that extends any existing on- line storage service. The architecture presented is both space-efficient (the user stores only a single hash value) and time efficient (a very small overhead is added to the operations of the storage service). The implementation was built on top of Amazons S3 and EC2 services. The experimental results confirm the negligible time overhead and scalability of our service.

Another important issue covered in this dissertation was the security usability of outsourced services. We have presented an effective method to visualize file system access control. We have outlined the design of a tool that visualizes both effective and local permissions and inheritance interruption for the Windows NTFS file system. An extensive user study was presented that showed the usability improvement of this tool in respect to Windows standard tool.

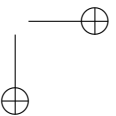
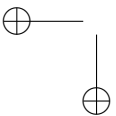
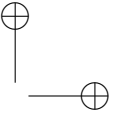
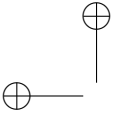
## Future Directions

In the future we would like to investigate how to authenticate more complex queries making use of a larger set of relational operations. Further, we would like to study models to build integrity verification services for collaborative environments like *Wikipedia*. In particular we are interested in the authentication of aggregation queries like *sum*, *average*, *max*, *min* and others.

We would like also to investigate the possibility to apply the authentication techniques proper of the *Authenticated Data Structures* to tags that allow the identification through *Radio Frequency*, the *RFID* technology. In particular we would like to authenticate the supply chain of objects at different level of aggregation for instance: the single item, an entire pallet and the container.

On the visualization topic work already in progress is the implementation of a full prototype of our system and to perform more user studies to evaluate our approach. As future work, we plan to develop further variations of the treemap layout to display additional file permission information also in network file system.

# Appendices



## TrACE User Study

The user study was conducted using the questionnaire that is reported integrally in the next two pages of this section together with a snapshot of TrACE tool that is required to answer questions in usage section.

Further, this survey was authorized by Brown University Institutional Review Board (IRB) on behalf of our protocol that follows the Brown University Research Privacy Policy. The full protocol is reported in this section, too.

## TRACE QUESTIONNAIRE

Please take a moment to fill out this anonymous survey. Your feedback is appreciated, and important to the continued improvement of TrACE, whose development at Brown University is supported by a research grant from the National Science Foundation (Roberto Tamassia, PI). The collected surveys will be aggregated and analyzed, and the results of this user study will be published in a research paper.

### BACKGROUND

1. Which operating system do you use primarily?

☐ Linux ☐ Mac OS ☐ Windows ☐ Other: .....

2. How often do you **inspect** file permissions?

☐ Once a week or more ☐ Once a month ☐ Once a year ☐ Never

3. How often do you **change** file permissions?

☐ Once a week or more ☐ Once a month ☐ Once a year ☐ Never

4. How familiar are you with the NTFS access control schema?

☐ Not at all ☐ Somewhat ☐ Quite ☐ Very

### USAGE (refer to the screenshot)

1. Is the **ReadOnly** permission of file **EXCEL.exe** in **Charlie** folder inherited?

☐ Yes ☐ No ☐ It is impossible to say

2. If yes, where did the permission originate? Check all that apply.

☐ Home ☐ Charlie ☐ QuickLinks ☐ It is impossible to say

3. If we change the permissions on the **Expense Reports** folder in **Bob** folder from **ReadOnly** to **ReadAndWrite**, the permissions of the file **September.txt** will be:

☐ More restrictive ☐ More permissive ☐ Unaffected ☐ It is impossible to say

4. Why does file **September.txt** have a different color than its parent folder **Expense Reports**? Check all that apply.

☐ There is a break of inheritance set on folder **Expense Reports**  
☐ There are additional permissions set explicitly on file **September.txt**  
☐ Some of the permissions of folder **Expense Reports** are set not to propagate to its children

### DESIGN

1. The color green is a good choice to indicate fewer privileges, because it is safer for the administrator.

<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> Disagree	<input type="checkbox"/> Neither agree nor disagree	<input type="checkbox"/> Agree	<input type="checkbox"/> Strongly agree
--	-----------------------------------	---	--------------------------------	---

2. The color red is a good choice to indicate fewer privileges, because it is more restrictive for the user.

<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> Disagree	<input type="checkbox"/> Neither agree nor disagree	<input type="checkbox"/> Agree	<input type="checkbox"/> Strongly agree
--	-----------------------------------	---	--------------------------------	---

3. The “baseline” display scheme is intuitive.

<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> Disagree	<input type="checkbox"/> Neither agree nor disagree	<input type="checkbox"/> Agree	<input type="checkbox"/> Strongly agree
--	-----------------------------------	---	--------------------------------	---

4. The “full spectrum” display is more useful than “baseline” in locating incorrect access control configurations.

<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> Disagree	<input type="checkbox"/> Neither agree nor disagree	<input type="checkbox"/> Agree	<input type="checkbox"/> Strongly agree
--	-----------------------------------	---	--------------------------------	---

5. Which of the two display schemes — “full spectrum” or “baseline” — would you be likely to use most often?

<input type="checkbox"/> Full Spectrum	<input type="checkbox"/> Baseline
--	-----------------------------------

6. If a visualization feature like TrACE was integrated into Windows File Explorer, a user could better understand and manage file system permissions.

<input type="checkbox"/> Strongly disagree	<input type="checkbox"/> Disagree	<input type="checkbox"/> Neither agree nor disagree	<input type="checkbox"/> Agree	<input type="checkbox"/> Strongly agree
--	-----------------------------------	---	--------------------------------	---

7. Other comments? Questions? Suggestions?

Thanks for your time!

## TRACE — Treemap Access Control Evaluator

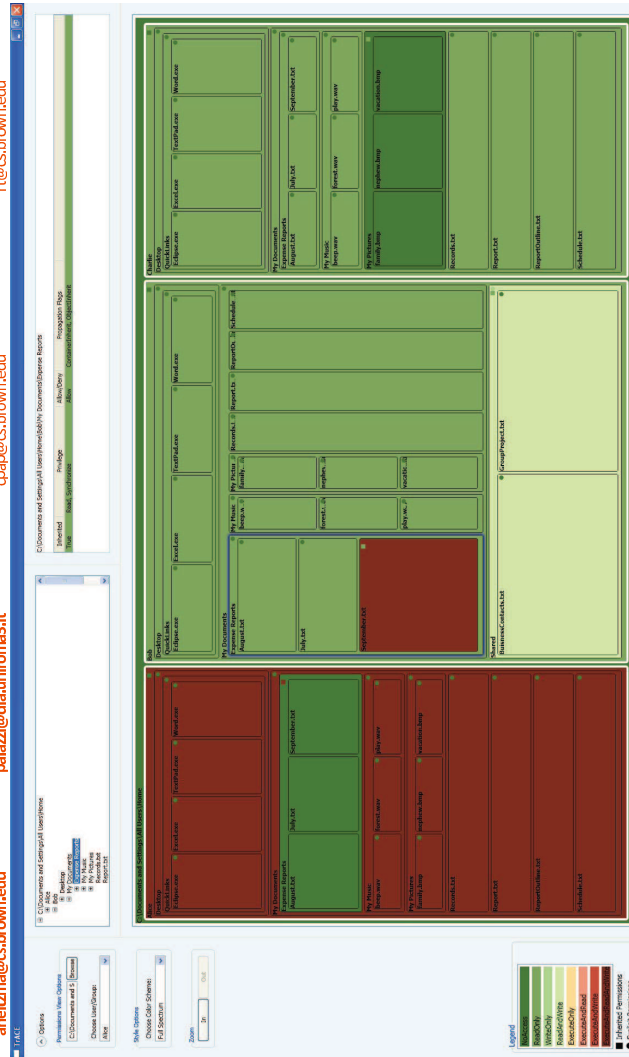
A visualization tool to aid in the analysis and management of file system permissions.

Alexander Heitzmann  
aheitzma@cs.brown.edu

Bernardo Palazzi  
palazzi@dia.uniroma3.it

Charalampos Papamantou  
cpap@cs.brown.edu

Roberto Tamassia  
rt@cs.brown.edu



Sponsors:  Effective Visualization of File System Access Control, VizSEC 2008

## Effective Computer Security Visualization

### *Additional Protocol Components*

**Roberto Tamassia, PI**

**September 10, 2008**

The protocol described in this document is part of an NSF-sponsored research effort aimed at developing graphical techniques and system prototypes for the visualization of computer security concepts.

We are building a tool called TrACE that helps computer users (both administrators and end users) analyze and understand the security configuration of their system. TrACE provides visual representations because they are more effective than textual information, such as activity logs or lists of authorizations. The visualization of access control permissions in a file system is the main component of TrACE. Indeed, as storage costs drop, users are increasingly managing large collections of files and find it difficult to set the correct permission to share document only with the intended people within their organization.

We plan to perform the following studies to evaluate the effectiveness and usability of TrACE.

#### **1. Analysis of anonymized course surveys and assignments.**

Visualizations produced by TrACE will be used as instructional materials in regular courses offered by the CS department, including CSCI1660 (computer security), CSCI1670/1690 (operating systems), and CSCI2520 (computational geometry). Anonymized survey and assignment data from these courses related to TrACE will be analyzed by the investigators to assess the effectiveness of TrACE. Aggregated results without individually identifiable information will be included in project-related publications.

- The first method for collecting assignment and survey data is described in the project proposal (see Section 1.3 of the attached document “Evaluation and Research Plan”). This method was previously reviewed and it was determined that it does not involve human subjects research (see the attached letter dated May 24, 2007).
- The second method for collecting assignment and survey data is similar to the first method except that it involves the participation of the investigators in (1) presenting TrACE to the students by giving (portions of) lectures and help sessions; (2) participating in the administration of class surveys and assignments related to TrACE; and (3) anonymizing the collected data.

#### **2. Analysis of anonymous surveys conducted at conferences.**

Surveys will be conducted during conferences and workshops where the investigators will give talks, demos and poster presentations about TrACE, including:

- Workshop on Visualization for Cyber Security (VizSec) and International Symposium on Recent Advances in Intrusion Detection (RAID ), September 15, 2008 Tang Center, MIT, Cambridge, MA , <http://www.vizsec.org/workshop2008/program.html>

- International Symposium on Graph Drawing (GD), September 21-24, 2008, Heraklion, Crete, Greece, [http://www.gd2008.org/docs/conf\\_program.pdf](http://www.gd2008.org/docs/conf_program.pdf)
- International Exhibition of Information & Communications Technology (SMAU), October 15-18, 2008, Milan, Italy

Attendees of these presentations will be given the opportunity to participate in a survey by anonymously filling out a questionnaire about the design and usability of TrACE (example attached). The questionnaires, which do not include individually identifiable information about the attendees, will be collected and analyzed by the investigators and aggregated results will be included in project-related publications.

## Bibliography

- [1] Amazon S3 (simple storage service). <http://aws.amazon.com/s3>.
- [2] JetS3t, an open source java toolkit for amazon s3. <http://jets3t.s3.amazonaws.com/index.html>.
- [3] Aris Anagnostopoulos, Michael T. Goodrich, and Roberto Tamassia. Persistent authenticated dictionaries and their applications. In *Proc. Information Security Conference (ISC 2001)*, volume 2200 of *LNCIS*, pages 379–393. Springer-Verlag, 2001.
- [4] A. Asuncion and D.J. Newman. UCI machine learning repository, University of California, Irvine, School of Information and Computer Sciences, 2007.
- [5] Giuseppe Ateniese, Randal Burns, Reza Curtmola, Joseph Herring, Lea Kissner, Zachary Peterson, and Dawn Song. Provable data possession at untrusted stores. In *Proc. Computer and Communication Security (CCS)*, 2007.
- [6] Robert Ball, Glenn A. Fink, and Chris North. Home-centric visualization of network traffic for security administration. In *VizSEC/DMSEC '04: Proceedings of the 2004 ACM workshop on Visualization and data mining for computer security*, pages 55–64, New York, NY, USA, 2004. ACM.
- [7] Naser S. Barghouti, John Mocenigo, and Wenke Lee. *Grappa*: A GRAPh PAKage in Java. In G. Di Battista, editor, *Graph Drawing (Proc. GD '97)*, volume 1353 of *Lecture Notes Comput. Sci.*, pages 336–343. Springer-Verlag, 1997.
- [8] Web based Database Software Solutions On-Demand. <http://www.teamdesk.net>.

- [9] Giuseppe Di Battista, Federico Mariani, Maurizio Patrignani, and Maurizio Pizzonia. Bgplay: A system for visualizing the interdomain routing evolution. In *Graph Drawing*, pages 295–306, 2003.
- [10] Thomas Bladh, David A. Carr, and Jeremiah Scholl. Extending tree-maps to three dimensions: A comparative study. In *Proc. Int. Conf. on Human Computer Interaction (HCI)*, pages 50–59, 2004.
- [11] M. Bruls, K. Huizing, and J. van Wijk. Squarified treemaps. In *Proc. of Joint Eurographics and IEEE TCVG Symp. on Visualization (TCVG)*, pages 33–42, 2000.
- [12] A. Buldas, M. Roos, and J. Willemson. Undeniable replies for database queries. In *In Proceedings of the Fifth International Baltic Conference on DB and IS, volume 2, pages 215-226, 2002.*, 2002.
- [13] Ahto Buldas, Peeter Laud, and Helger Lipmaa. Accountable certificate management using undeniable attestations. In *CCS '00: Proceedings of the 7th ACM conference on Computer and communications security*, pages 9–17, New York, NY, USA, 2000. ACM.
- [14] Jan Camenisch and Anna Lysyanskaya. Dynamic accumulators and application to efficient revocation of anonymous credentials. In *Proc. CRYPTO*, 2002.
- [15] Xiang Cao and Lee Iverson. Intentional access management: making access control usable for end-users. In *Proc. of Int. Symposium on Usable Privacy and Security (SOUPS)*, pages 20–31, 2006.
- [16] Joe Celko. *Joe Celko's Trees and hierarchies in SQL for smarties*. Morgan-Kaufmann, 2004.
- [17] Matthew Chalmers. A linear iteration time layout algorithm for visualising high-dimensional data. In *VIS '96: Proceedings of the 7th conference on Visualization '96*, pages 127–ff., Los Alamitos, CA, USA, 1996. IEEE Computer Society Press.
- [18] Greg Conti. *Security Data Visualization*. No Starch Press, San Francisco, CA, USA, 2007.
- [19] C.J. Date. Why is it so difficult to provide a relational interface to ims. In *Relational Database— Selected Writings*, pages 241–257. Addison-Wesley, 1986.

## BIBLIOGRAPHY

131

- [20] Prem Devanbu, Michael Gertz, April Kwong, Chip Martel, Glen Nuckolls, and Stuart G. Stubblebine. Abstract flexible authentication of xml documents.
- [21] Premkumar Devanbu, Michael Gertz, April Kwong, Chip Martel, Glen Nuckolls, and Stuart Stubblebine. Flexible authentication of XML documents. *Journal of Computer Security*, 6:841–864, 2004.
- [22] Premkumar Devanbu, Michael Gertz, Chip Martel, and Stuart G. Stubblebine. Authentic third-party data publication. In *Fourteenth IFIP 11.3 Conference on Database Security*, 2000.
- [23] Giuseppe Di Battista and Bernardo Palazzi. Authenticated relational tables and authenticated skip lists. In *Proc. Working Conf. on Data and Applications Security (DBSEC)*, pages 31–46, 2007.
- [24] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, Gerardo Pelosi, and Pierangela Samarati. Preserving confidentiality of security policies in data outsourcing. In *WPES '08: Proceedings of the 7th ACM workshop on Privacy in the electronic society*, pages 75–84, New York, NY, USA, 2008. ACM.
- [25] Sabrina De Capitani di Vimercati, Sara Foresti, Sushil Jajodia, Stefano Paraboschi, and Pierangela Samarati. A data outsourcing architecture combining cryptography and access control. In *CSAW '07: Proceedings of the 2007 ACM workshop on Computer security architecture*, pages 63–69, New York, NY, USA, 2007. ACM.
- [26] P. Eades. A heuristic for graph drawing. *Congr. Numer.*, 42:149–160, 1984.
- [27] J. Ellson, E. R. Gansner, L. Koutsofios, S. C. North, and G. Woodhull. Graphviz and dynagraph - static and dynamic graph drawing tools. *Graph Drawing Software*, 2003.
- [28] Joshua Foster, Kalpathi Subramanian, Robert Herring, and Gail Ahn. Interactive exploration of the AFS file system. In *Proc. of the IEEE Symposium on Information Visualization (INFOVIS)*, page 215, 2004.
- [29] T. Fruchterman and E. Reingold. Graph drawing by force-directed placement. *Softw. – Pract. Exp.*, 21(11):1129–1164, 1991.

- [30] Kevin Fu, M. Frans Kaashoek, and David Mazieres. Fast and secure distributed read-only file system. *Computer Systems*, 20(1):1–24, 2002.
- [31] Irene Gassko, Peter S. Gemmell, and Philip MacKenzie. Efficient and fresh certification. In *Int. Workshop on Practice and Theory in Public Key Cryptography (PKC ’2000)*, volume 1751 of *LNCS*, pages 342–353. Springer-Verlag, 2000.
- [32] Luc Girardin and Dominique Brodbeck. A visual approach for monitoring logs. In *LISA ’98: Proceedings of the 12th USENIX conference on System administration*, pages 299–308, Berkeley, CA, USA, 1998. USENIX Association.
- [33] Eu-Jin Goh, Hovav Shacham, Nagendra Modadugu, and Dan Boneh. SiRiUS: securing remote untrusted storage. In *Proc. Network and Distributed System Security Symp. (NDSS)*, pages 131–145, 2003.
- [34] M. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical report, Johns Hopkins Information, 2000.
- [35] M. T. Goodrich, C. Papamanthou, R. Tamassia, and N. Triandopoulos. Athos: Efficient authentication of outsourced file systems. In *Proc. Information Security Conference*, *LNCS*, pages 80–96. Springer, 2008.
- [36] M. T. Goodrich and R. Tamassia. Efficient authenticated dictionaries with skip lists and commutative hashing. Technical report, Johns Hopkins Information Security Institute, 2000. Available from <http://www.cs.brown.edu/cgc/stms/papers/hashskip.pdf>.
- [37] Michael T. Goodrich, James Lentini, Michael Shin, Roberto Tamassia, and Robert Cohen. Design and implementation of a distributed authenticated dictionary and its applications. Technical report, Center for Geometric Computing, Brown University, 2002. Available from <http://www.cs.brown.edu/cgc/stms/papers/stms.pdf>.
- [38] Michael T. Goodrich, Michael Shin, Roberto Tamassia, and William H. Winsborough. Authenticated dictionaries for fresh attribute credentials. In *Proc. Trust Management Conference*, volume 2692 of *LNCS*, pages 332–347. Springer, 2003.

## BIBLIOGRAPHY

133

- [39] Michael T. Goodrich and Roberto Tamassia. *Data Structures and Algorithms in Java*. John Wiley & Sons, Inc., New York, NY, USA, 2000.
- [40] Michael T. Goodrich, Roberto Tamassia, and Jasminka Hasic. An efficient dynamic and distributed cryptographic accumulator. In *Proc. of Information Security Conference (ISC)*, volume 2433 of *LNCS*, pages 372–388. Springer-Verlag, 2002.
- [41] Michael T. Goodrich, Roberto Tamassia, and Andrew Schwerin. Implementation of an authenticated dictionary with skip lists and commutative hashing. In *Proc. 2001 DARPA Information Survivability Conference and Exposition*, volume 2, pages 68–82, 2001.
- [42] Michael T. Goodrich, Roberto Tamassia, Nikos Tri, and Oupoulos Robert Cohen. Authenticated data structures for graph and geometric searching. In *in CT-RSA*, pages 295–313. Springer, LNCS, 2003.
- [43] A. Heitzmann, B. Palazzi, C. Papamanthou, and R. Tamassia. Effective visualization of file system access-control. In *Proc. Int. Workshop on Visualization for Cyber Security (VizSec)*, volume 5210 of *LNCS*, pages 18–25. Springer, 2008.
- [44] Alexander Heitzmann, Bernardo Palazzi, Charalampos Papamanthou, and Roberto Tamassia. Efficient integrity checking of untrusted network storage. In *StorageSS '08: Proceedings of the 4th ACM international workshop on Storage security and survivability*, pages 43–54, New York, NY, USA, 2008. ACM.
- [45] Trent Jaeger and Jonathon E. Tidswell. Practical safety in flexible access control models. *ACM Trans. Information Systems Security*, 4(2):158–190, 2001.
- [46] Ravi Chandra Jammalamadaka, Roberto Gamboni, Sharad Mehrotra, Kent E. Seamons, and Nalini Venkatasubramanian. gVault: A gmail based cryptographic network file system. In *Proc. Working Conf. on Data and Applications Security (DBSEC)*, pages 161–176, 2007.
- [47] B. Johnson and Ben Shneiderman. Tree maps: A space-filling approach to the visualization of hierarchical information structures. In *Proc. IEEE Visualization*, pages 284–291, 1991.

- [48] John B. Johnston. The contour model of block structured processes. *SIGPLAN Not.*, 6(2):55–82, 1971.
- [49] Vishal Kher and Yongdae Kim. Securing distributed storage: challenges, techniques, and systems. In *Proc. ACM Workshop on Storage Security and Survivability*, pages 9–25, 2005.
- [50] P. C. Kocher. On certificate revocation and validation. In *Proc. Int. Conf. on Financial Cryptography*, volume 1465 of *LNCS*. Springer-Verlag, 1998.
- [51] Paul C. Kocher. On certificate revocation and validation. In *FC ’98: Proceedings of the Second International Conference on Financial Cryptography*, pages 172–177, London, UK, 1998. Springer-Verlag.
- [52] Ramakrishna Kotla, Lorenzo Alvisi, and Michael Dahlin. Safestore: A durable and practical storage system. In *USENIX Annual Technical Conference*, pages 129–142. USENIX, 2007.
- [53] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *Proc. of ACM SIGMOD International Conference on Management of Data*, pages 121–132, 2006.
- [54] F. Li, M. Hadjieleftheriou, G. Kollios, and L. Reyzin. Dynamic authenticated index structures for outsourced databases. In *SIGMOD ’06: Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, pages 121–132, New York, NY, USA, 2006. ACM Press.
- [55] Jinyuan Li, Maxwell N. Krohn, David Mazières, and Dennis Shasha. Secure untrusted data repository (SUNDR). In *OSDI*, pages 121–136, 2004.
- [56] Umesh Maheshwari, Radek Vingralek, and William Shapiro. How to build a trusted database system on untrusted storage. In *Proc. USENIX Symp. on Operating Systems Design and Implementation*, 2000.
- [57] James Manger. Response on Jungle Disk Blog. <http://blog.jungledisk.com/2006/06/06/encryption/#comment-26>.
- [58] Petros Maniatis and Mary Baker. Enabling the archival storage of signed documents. In *Proc. USENIX Conf. on File and Storage Technologies (FAST 2002)*, Monterey, CA, USA, 2002.

## BIBLIOGRAPHY

135

- [59] Petros Maniatis and Mary Baker. Secure history preservation through timeline entanglement. In *Proc. USENIX Security Symposium*, 2002.
- [60] Florian Mansmann, Lorenz Meier, and Daniel Keim. Graph-based monitoring of host behavior for network security. In *Proc. Workshop on Visualization for Computer Security (VizSec)*, 2007.
- [61] C. Martel, G. Nuckolls, P. Devanbu, M. Gertz, A. Kwong, and S. G. Stubblebine. A general model for authenticated data structures. *Algorithmica*, 39(1):21–41, 2004.
- [62] Chip Martel, Glen Nuckolls, Premkumar Devanbu, Michael Gertz, April Kwong, and Stuart Stubblebine. A general model for authentic data publication, 2001. Available from <http://www.cs.ucdavis.edu/~devanbu/files/model-paper.pdf>.
- [63] A. Meier, R. Dippold, J. Mercerat, A. Muriset, J. Untersinger, R. Eckerlin, and F. Ferrara. Hierarchical to relational database migration. *IEEE Softw.*, 11(3):21–27, 1994.
- [64] R. C. Merkle. Protocols for public key cryptosystems. In *Proc. Symp. on Security and Privacy*, pages 122–134. IEEE Computer Society Press, 1980.
- [65] R. C. Merkle. A certified digital signature. *Advances in Cryptology-Crypto’89*, 435:218–238, 1989.
- [66] Ralph C. Merkle. A certified digital signature. In G. Brassard, editor, *Proc. CRYPTO ’89*, volume 435 of *LNCS*, pages 218–238. Springer-Verlag, 1989.
- [67] Gerome Miklau and Dan Suciu. Implementing a tamper-evident database system. In *Data Management on the Web, Proc. Asian Computing Science Conf.*, volume 3818 of *LNCS*, pages 28–48. Springer, 2005.
- [68] Gerome Miklau and Dan Suciu. Implementing a tamper-evident database system. In *ASIAN: 10th Asian Computing Science Conference*, pages 28–48, 2005.
- [69] Jaime Montemayor, Andrew Freeman, John Gersh, Thomas Llanso, and Dennis Patrone. Information visualization for rule-based resource access control. In *Proc. of Int. Symposium on Usable Privacy and Security (SOUPS)*, 2006.

- [70] Chris Muelder, Kwan-Liu Ma, and Tony Bartoletti. A visualization methodology for characterization of network scans. In *VIZSEC '05: Proceedings of the IEEE Workshops on Visualization for Computer Security*, page 4, Washington, DC, USA, 2005. IEEE Computer Society.
- [71] J. Ian Munro, T. Papadakis, and R. Sedgewick. Deterministic skip lists. In *SODA '92: Proceedings of the third annual ACM-SIAM symposium on Discrete algorithms*, pages 367–375, Philadelphia, PA, USA, 1992.
- [72] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, 2006.
- [73] Einar Mykletun, Maithili Narasimha, and Gene Tsudik. Authentication and integrity in outsourced databases. *Trans. Storage*, 2(2):107–138, 2006.
- [74] Moni Naor and Kobbi Nissim. Certificate revocation and certificate update. In *Proc. 7th USENIX Security Symposium*, pages 217–228, Berkeley, 1998.
- [75] Maithili Narasimha and Gene Tsudik. Authentication of outsourced databases using signature aggregation and chaining. In *International Conference on Database Systems for Advanced Applications (DASFAA)*. DASFAA, 2006.
- [76] A. Noack. An energy model for visual graph clustering. 2003.
- [77] S. Noel, M. Jacobs, P. Kalapa, and S. Jajodia. Multiple coordinated views for network attack graphs. In *Proc. IEEE Workshop on Visualization for Computer Security (VizSEC)*, pages 99–106, 2005.
- [78] Steven Noel and Sushil Jajodia. Managing attack graph complexity through visual hierarchical aggregation. In *CCS Workshop on Visualization and Data Mining for Computer Security*, 2004.
- [79] Jon Oberheide, Manish Karir, and Dionysus Blazakis. Vast: visualizing autonomous system topology. In *VizSEC '06: Proceedings of the 3rd international workshop on Visualization for computer security*, pages 71–80, New York, NY, USA, 2006. ACM.
- [80] Caspio Bridge online database. <http://www.caspio.com>.

## BIBLIOGRAPHY

137

- [81] Alina Oprea and Micheal K. Reiter. Integrity checking in cryptographic file systems with constant trusted storage. In *Proc. USENIX Security Symposium (USENIX)*, pages 183–198, 2007.
- [82] H. Pang, A. Jain, K. Ramamritham, and K. Tan. Verifying completeness of relational query results in data publishing. In *SIGMOD Conference*, pages 407–418, 2005.
- [83] H. Pang and K. Tan. Authenticating query results in edge computing. In *ICDE '04: Proceedings of the 20th International Conference on Data Engineering*, page 560, Washington, DC, USA, 2004. IEEE Computer Society.
- [84] C. Papamanthou and R. Tamassia. Time and space efficient algorithms for two-party authenticated data structures., 2007.
- [85] Daniel J. Polivy and Roberto Tamassia. Authenticating distributed data using Web services and XML signatures. In *Proc. ACM Workshop on XML Security*, 2002.
- [86] Livebase project Blog on Web-based db. <http://livebase.blog.com/1142527/>.
- [87] William Pugh. Skip lists: A probabilistic alternative to balanced trees. In *Workshop on Algorithms and Data Structures*, pages 437–449, 1989.
- [88] Jennifer Rode Carolina Johansson Paul DiGioia Roberto Silveira Silva Filho Kari Nies David H. Nguyen Jie Ren Paul Dourish David F. Redmiles. Seeing further: extending visualization as a basis for usable security. In *SOUPS*, pages 145–155, 2006.
- [89] R.W Reeder, L. Bauer, L.F. Cranor, M.K. Reiter, K. Bacon, K. How, and H. Strong. Expandable grids for visualizing and authoring computer security policies. In *Proc. ACM Conf. on Human Factors in Computing Systems (CHI)*, pages 1473–1482, 2008.
- [90] Mark E. Russinovich and David A. Solomon. *Microsoft Windows Internals, Fourth Edition: Microsoft Windows Server <sup>TM</sup>2003, Windows XP, and Windows 2000 (Pro-Developer)*. Microsoft Press, Redmond, WA, USA, 2004.

- [91] Thomas S. J. Schwarz and Ethan L. Miller. Store, forget, and check: Using algebraic signatures to check remotely administered storage. In *ICDCS '06: IEEE Int. Conf. on Distributed Computing Systems*, page 12, 2006.
- [92] P. Griffiths Selinger, M. M. Astrahan, D. D. Chamberlin, R. A. Lorie, and T. G. Price. Access path selection in a relational database management system. In *SIGMOD '79: Proceedings of the 1979 ACM SIGMOD international conference on Management of data*, pages 23–34, New York, NY, USA, 1979. ACM.
- [93] Hovav Shacham and Brent Waters. Compact proofs of retrievability. Cryptology ePrint Archive, 08/073, 2008.
- [94] Michael Shin, Christian Straub, Roberto Tamassia, and Daniel J. Polivy. Authenticating Web content with prooflets. Technical report, Center for Geometric Computing, Brown University, 2002. <http://www.cs.brown.edu/cgc/stms/papers/prooflets.pdf>.
- [95] R. Sion. Query execution assurance for outsourced databases. In *VLDB '05*, pages 601–612. VLDB Endowment, 2005.
- [96] John Stasko. An evaluation of space-filling information visualizations for depicting hierarchical structures. *Int. J. Hum.-Comput. Stud.*, 53(5):663–694, 2000.
- [97] R. Tamassia, B. Palazzi, and C. Papamanthou. Graph drawing for security visualization. In Maurizio Patrignani, editor, *Graph Drawing (Proc. GD '08)*, Lecture Notes Comput. Sci. Springer-Verlag, 2008.
- [98] Roberto Tamassia. Authenticated data structures. In *Proc. European Symp. on Algorithms*, volume 2832 of *LNCS*, pages 2–5. Springer-Verlag, 2003.
- [99] Roberto Tamassia and Nikos Triandopoulos. On the cost of authenticated data structures. Technical report, Center for Geometric Computing, Brown University, 2003. Available from <http://www.cs.brown.edu/cgc/stms/papers/costauth.pdf>.
- [100] Roberto Tamassia and Nikos Triandopoulos. Computational bounds on hierarchical data processing with applications to information security. In *Proc. Int. Colloquium on Automata, Languages and Programming (ICALP)*, volume 3580 of *LNCS*, pages 153–165. Springer-Verlag, 2005.

## BIBLIOGRAPHY

139

- [101] Soon Tee Teoh, Supranamaya Ranjan, Antonio Nucci, and Chen-Nee Chuah. Bgp eye: a new visualization tool for real-time detection and analysis of bgp anomalies. In *VizSEC '06: Proceedings of the 3rd international workshop on Visualization for computer security*, pages 81–90, New York, NY, USA, 2006. ACM.
- [102] Juan Toledo. Etherape a live graphical network monitor tool. <http://etherape.sourceforge.net/>.
- [103] Jens Tölle and Oliver Niggemann. Supporting intrusion detection by graph clustering and graph drawing. In *Proc. of Third International Workshop on Recent Advances in Intrusion Detection (RAID 2000)*, Toulouse, France, 2000.
- [104] Richard M. Wilson and R. Daniel Bergeron. Dynamic hierarchy specification and visualization. In *Proc. of the IEEE Symposium on Information Visualization (INFOVIS)*, page 65, 1999.
- [105] Danfeng Yao, Michael Shin, Roberto Tamassia, and William H. Winsborough. Visualization of automated trust negotiation. In *VIZSEC '05: Proceedings of the IEEE Workshops on Visualization for Computer Security*, page 8, Washington, DC, USA, 2005. IEEE Computer Society.
- [106] Xiaoxin Yin, William Yurcik, Michael Treaster, Yifan Li, and Kiran Lakkaraju. VisFlowConnect: Netflow visualizations of link relationships for security situational awareness. In *Proc. ACM Workshop on Visualization and Data Mining for Computer Security (VizSEC/DMSEC)*, pages 26–34, New York, NY, USA, 2004. ACM Press.
- [107] Aydan Y. Yumerefendi and Jeffrey S. Chase. Strong accountability for network storage. In *Proc. Conference on File and Storage Technologies (FAST)*, pages 77–92, 2007.
- [108] online database Zoho Creator. <http://creator.zoho.com>.