SEALED:

SEARCHING ENCRYPTED AUDIT LOGS EXPEDITIOUSLY

by
Darren Craig Davis

A thesis submitted to The Johns Hopkins University in conformity with the requirements for the degree of Master of Science in Engineering

Baltimore, Maryland
May 2004

© Darren Craig Davis 2004
All Rights Reserved

Abstract

In this thesis, we explore restricted delegation of searches on encrypted audit logs. We show how to limit the exposure of private information stored in the log during such a search and provide a technique to delegate searches on the log to an investigator. These delegated searches are limited to authorized keywords that pertain to specific time periods, and provide guarantees of completeness to the investigator. Moreover, we show that investigators can efficiently find all relevant records, and can authenticate retrieved records without interacting with the owner of the log. In addition, we provide an empirical evaluation of our techniques using encrypted logs comprising of approximately $27,000$ records of IDS alerts collected over a span of a few months.

Advisor: Fabian Monrose
Assistant Professor
Security and Privacy Applied Research Lab
Department of Computer Science
Johns Hopkins University

Acknowledgments

I am indebted to my parents for their unwavering support and encouragement. They exposed me to computers when I was young, which started my interest in Computer Science. Without their support I would not have been able to attend Johns Hopkins University. My experiences here will affect me for the rest of my life, and for this, I will always be grateful.

I would like to thank my advisor, Professor Fabian Monrose, for his guidance, encouragement, and friendship. His many suggestions and revisions have largely helped this thesis. I am also grateful for the support of Professor Michael K. Reiter at Carnegie Mellon University for his interest in this work. Professor Reiter provided numerous helpful suggestions that improved earlier versions of my thesis.

I would also like to thank everyone in the Security and Privacy Applied Research Lab for being there to bounce ideas off of, and for their useful feedback. In particular, I would like to thank Seny Kamara for pre-processing the Snort logs used in my experiments. Moreover, I would like to thank the Information Security Institute for supporting this work.

Finally, I would like to thank my friends, particularly Michael Hilsdale, Raphael Schweber-Koren, Ashima Munjal, Peter Keeler, and Ryan Caudy, for dragging me out to have some fun every once in a while.

Darren Davis

May 7, 2004

Contents

1 Introduction **1**

2 Related Work **6**

3 Settings and Goals **12**

3.1 Assumptions 14

3.2 Variables, Functions, and Notation 15

3.3 Investigators and Keys 16

3.4 Investigators and Searches 19

3.5 Recovery from Compromise 20

4 Preliminaries **23**

4.1 Cryptographic Primitives 23

4.2 Tamper Resistant Audit Logs 24

4.3 IBE-Enabled Audit Logs 26

5 Time Zones **30**

5.1 Time Periods 31

5.2 A Trivial Solution 32
5.3 A Tree Based Approach 33
5.4 Per Keyword Time Zones 35
5.5 A Single Time Zone 37

6 An Index for Efficient Searching **40**
6.1 A Separate Index 41
6.2 An Integrated Index 42
6.3 Anchor Records 43

7 SEALED **45**
7.1 Normal Log Records 46
7.2 Anchor Records 47
7.3 Protocols 49
7.3.1 SETUP 49
7.3.2 DELEGATION 49
7.3.3 SEARCHING AND DECRYPTION 50

8 Performance **54**

9 Conclusion and Future Work **58**

A Algorithm Details **60**
A.1 LOG 60
A.2 DELEGATE 60
A.3 SEARCH 65

List of Tables

8.1 Log Import Timings 56

List of Figures

1.1 Log Structure Overview . . . 5

4.1 Original Indexing Scheme . . . 29

5.1 Time Period Bit Structure . . . 31

5.2 Tree Delegation Overview . . . 34

7.1 Searching In SEALED . . . 51

8.1 Searching Performance . . . 56

A.1 Time Key Distribution . . . 63

List of Algorithms

1 Message Insertion 61

2 Time Zone Advancement 62

3 Distribution of Time Keys 64

4 Investigator Search Engine (Phase 1) 66

5 Investigator Search Engine (Phase 2) 67

Chapter 1

Introduction

Databases contain large amounts of sensitive information, and as such, are tempting targets for adversaries. In general, the company that owns the database considers the information stored within it to be a valuable asset worth protecting. For example, many web sites have privacy policies that describe how they collect users' information, and how they protect it from disclosure. However, in today's competitive business climate, security and privacy tend to be at a lower priority than operational cost. Subcontracting, and more recently outsourcing, have allowed sensitive data to be accessed by third parties whose primary concern may not be in upholding the reputation of the company outsourcing its data. For example, last year a medical transcriber in Pakistan hired by a subcontractor of UCSF Medical Center threatened to disclose patients' medical records unless financial incentives were received [L03]. Thus, as a result of increased outsourcing, people's private information now face more significant threats from both insiders and outsiders.

At a high level, the methods for securing the information residing in databases are dependent on the assumptions about adversary's limitations and more importantly, on the database's mode of operation. For example, a system may permit only a few users to update records, but allow nu-

merous users to have read-only access. Moreover, the nature of updates themselves may also differ across systems. In most systems, records may be updated or deleted after they are initially created; however, append-only settings exist where records cannot be altered after their initial creation, for example, medical records and audit logs. Moreover, the server hosting the database is usually trusted to perform the protocols specified, through it is typically assumed that such servers are susceptible to compromise.

Systems in the real world always face attacks. In order to be useable, the system must achieve some security goals while remaining efficient. These goals include limiting how much information is leaked during an attack and guaranteeing that an attack can be detected after the fact. For a system to accomplish these objectives, careful consideration must be given to what secrets the server stores, and the related consequences if that information is leaked to an adversary. In general, this necessitates that a server must delete or evolve old secrets so that an adversary who compromises the server does not have access to old secrets.

When compared to an setting in which only one user may append records, security goals are harder to achieve in settings where many users are able to modified and append records. Thus, we limit the scope of our investigation to a setting in which only new records are appended to the database, and only one entity can append records. When that entity is the server itself, the database model becomes very similar to that of an audit log. Audit logs serve the purpose of providing evidence that private information was read or altered. The more detailed an audit log, the more useful it may be in determining the specifics of an attack, including its severity. However, if obtained by an adversary, audit logs can leak substantial amounts of private information (for example, users' access patterns on a web site). As a result, the information stored in detailed audit logs is often as sensitive as the data on the systems they are monitoring.

To that end, we investigate several questions. First, we explore how an audit log can be searched during an investigation after the server is compromised, and how to limit the exposure of private information stored in the log during that search. Next, we examine how the owner of the log, whom we refer to as Alice, can delegate searches on the log to authorized third parties efficiently. These delegations may be voluntary (for example, to a managed security company to investigate an attack) or involuntary (for example, to comply with a search warrant). Delegated searches are limited in scope and duration, provide a guarantee of completeness, and are efficient.

To understand how important but sensitive audit logs are, consider the case when a company has a website that keeps an audit log of all accesses to that web site, and the server that runs the web site is outsourced to a third party. Without protecting the audit log, users' browsing habits can easily be examined by that third party, or any adversary that compromises the server. Many websites have privacy policies that state they will disclose collected information in order to comply with law enforcement (for example, CNN[1]). In the event that a search warrant is issued, the company needs to provide access to the appropriate log entries to an investigator. In addition to a guarantee of completeness, the investigator needs to be able to complete the search through the audit log efficiently. A recent example emphasizes the importance of the system being efficient for searches: earlier this year, "After several hours of attempting to track down, inspect and audit the terabytes of data that [CIT Hosting hosts], the FBI determined that it was more efficient [from their point of view] to remove all of [CIT Hosting's] servers and transport them to the FBI local laboratories for inspection." [2]

While there are existing proposals (described in more detail in Chapters 2 and 4) that separately implement audit logs on untrusted servers, searches on encrypted data by keywords, and time-

[1] See CNN's privacy policy at http://www.cnn.com/privacy.html
[2] See http://www.carrierhotels.com/news/2004/Feb/19/fbi_shutters_web_host.shtml

based cryptographic primitives, our scheme is the first that integrates all of these constructions. Our solution is based on the idea of forward security, namely that secrets are evolved to limit the number of secrets that the server is required to store. The log's owner, Alice, has the ability to delegate a limited searching capability to an untrusted third party investigator. Moreover, the search is limited in scope to only certain keywords, and is limited to records that pertain to a specific time period. The investigator can efficiently find all relevant records (i.e., the investigator does not have to perform a linear search through the entire log), and can be convinced that there are no more records within the scope of the investigation. Furthermore, in an effort to verify the authenticity of all of the results, only a constant amount of data needs to be transmitted between Alice and the investigator.

We propose several new ideas which allow for an efficient implementation of these goals. Specifically, our construction uses the hash of the encryption key for a record as an index for retrieving that record. When the key used to encrypt a record is a hash of the record itself, knowledge of the hash of a record is sufficient to retrieve that record from the server, decrypt it, and verify its integrity, without giving an adversary who compromised the server means of doing the same. Furthermore, hashes of previous records are included in later records, forming chains of records. Therefore, only the final record of a chain needs to be authenticated in order to authenticate all of the records in the chain. As a result of this construction, we believe that we have the first audit log that does not require that records be stored in a deterministic linear order; records may be stored in a random order, and even periodically reshuffled without affecting the security or usability of the log. Figure 1.1 depicts our construction.

The rest of this thesis is organized as follows. An overview of related work is discussed in Chapter 2. Chapter 3 describes the assumptions and goals of the system. Notation, cryptographic building blocks, and more details about some related schemes are covered in Chapter 4. The basic

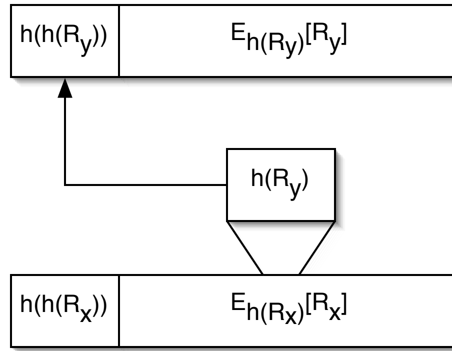

Figure 1.1: The database uses double hashes of log records (R_i's) to look up log records encrypted with their hashes, e.g. $E_{h(R_i)}[R_i]$. The encrypted record contains a hash of another record, in this example, R_y. Knowledge of $h(R_y)$ is sufficient to retrieve the encrypted R_y from the database, decrypt it, and verify its integrity. In addition, if R_x was authenticated, then R_y is authenticated.

ideas for time zones and building an index are presented in Chapters 5 and 6. Next, we present our scheme, SEALED, and performance results in Chapters 7 and 8. Finally, Chapter 9 concludes with a discussion of some possible optimizations and future work. Appendix A presents the specific algorithms used by SEALED.

Chapter 2

Related Work

There is much research that relates to the system discussed in this thesis. Work on outsourced databases, in particular, that of Mykletun, Narashimha and Tsudik [MNT04] propose a model for outsourced databases where a external server is trusted to store data, but is not trusted with the integrity and authenticity of that data. Within that model, they present a scheme which allows the server to efficiently prove the integrity and authenticity of data through the use of aggregate signatures[3]. However, confidentiality of the data is not considered, nor do they address the issue of ensuring that the server returns a complete answer. Moreover, the server is the only entity that performs queries, which may lead to a performance bottleneck. Our scheme, and many of the other schemes described in this chapter, allow searches to be delegated to third party investigators.

The topic of searches on encrypted data is directly related to audit log privacy. Schemes for searching encrypted data can be divided into proposals that use symmetric keys and ones that use asymmetric keys. First, we consider the schemes that use symmetric keys. In [SWP00], Song,

[3] The aggregate signature scheme they use was presented by Boneh et al in [BGL03]. The inverse of aggregating signatures is described in [JMS02], and can be used to redact part of a signature. A system that requires only part of a message to be authenticated and the rest of the message to remain undisclosed can take advantage of a redactable signature scheme.

Wagner and Perrig describe a scheme for searching encrypted documents stored on a server for any word in the document. They differ from our approach by having the server itself perform the search, and return the results to the document's owner. Their scheme does not allow searches to be delegated to third parties, and cannot detect a server misbehaving by omitting results (e.g. there is no guarantee of completeness), and requires a specific encryption scheme, rather than allowing any suitable encryption scheme from being used. Although symmetric key operations are efficient, the approach in [SWP00] still requires the server to perform a linear search through each document in the collection.

Several more recent papers use Bloom Filters [B70] to achieve more efficient searches (in models similar to that of [SWP00]), although a tradeoff of using Bloom Filters is false positives. One such scheme in this setting is presented by Goh in [G04]. A specially constructed Bloom Filter is sent with each encrypted (and compressed) document to be stored on a server. In addition, Goh describes how to efficiently search for infrequent words across multiple documents using a Bloom Filter Tree. A subsequent scheme by Bellovin and Cheswick [BC04] uses encrypted Bloom Filters to allow pairs of entities that do not trust each other to share selected data. A partially trusted third party can perform a key transformation to change a search under one entity's key to a search under another entity's key. However, if that third party colludes with any entities in the system, it learns the secret keys of all participating entities.

Other types of data structures may also be used to achieve fast searches. In [CM04], Chang and Mitzenmacher present a scheme to search encrypted files using a pre-generated index derived from keywords in each of these files. This scheme is able to avoid the problem of false positives that the previous schemes [SWP00, G04, BC04] suffer from. However, it may be argued that by introducing noise into a search's results, false positives provide a defense against statistical attacks

by an adversary. Thus, the cost of more efficient searches is that some information about access patterns is leaked. In addition, [CM04] requires the keywords used to index documents be fixed once an initial set of documents are indexed. As a result, there exist some keywords that cannot be used to index new documents. Without the ability to use any string as a keyword, searching may not be as precise as desired.

A symmetric key scheme for searching encrypted data is typically used by encrypting one's documents and sending them to a server that is able to search them, as is the case in the aforementioned approaches of [SWP00, G04, CM04]. However, in the context of an audit log, the server is the entity that creates and encrypts documents (log entries) that eventually need to be searched and decrypted by investigators. An example of a scheme in this setting was proposed in [WBD04] by Waters et. al.

A characteristic of existing schemes for searching encrypted data in a symmetric key setting is that the key used to encrypt records is also the key used to search and decrypt records. However, in the setting of an audit log, since the server encrypts log entries, it also has the keys to search and decrypt those entries. Thus, symmetric key schemes are not suitable for our purpose – once the server is compromised, an adversary would obtain all of the key material; as Waters et. al. point out, "... the scheme is insecure against an adversary that is able to compromise an audit log server." Thus, we require an asymmetric key scheme to achieve our goals.

Unfortunately, asymmetric key operations are more computationally expensive than symmetric ones, thus the ability to perform full-text searches in an asymmetric key setting may be computationally prohibitive. Solutions proposed in an asymmetric setting adopt the strategy of searching a limited number of keywords extracted each document. Provided that these keywords are carefully selected, this limitation should not limit useful searching (e.g., searching for the word "the" is not

useful).

The only works we are aware of about asymmetric schemes for searching encrypted data use the identity based encryption scheme of Boneh and Franklin [BF01] (more details about IBE are discussed later). In [BCO03], Boneh et. al. provide a formal definition for searchable public key-encryption, and provide three provably secure constructions. However, their constructions only allow for matching documents to be identified by the searcher; the searcher is unable to decrypted them. By contrast, in [WBD04], a novel construction using identity based encryption allows a searcher to find matching documents and decrypt them. Documents (specifically, log records) are encrypted with a randomly generated symmetric key, and that key is encrypted under each public key corresponding to a keyword in the document. This construction is used to build an searchable and encrypted audit log. Our scheme extends [WBD04] by increasing the efficiency of a search through the log and adding the ability to restrict searches to the portion of the log corresponding to a particular interval of time. Additional details about the scheme presented in [WBD04] are given in Section 4.3.

A more distantly related topic is Private Information Retrieval [CGK95], where the goal is to prevent a set of non-colluding servers from learning the physical location of a document (which can be thought of as a block of data) that is being retrieved from a database. Enhancements were presented in [CGN97], which described several data structures that can be used to determine the physical location of a document. Of particular interest here is construction where a document's physical location is a hash of the document itself. While this is not very useful for searching with the granularity less than an entire document, our system uses a similar construction for retrieving documents. In our case, a hash of a document's encryption key can be used to retrieve the document.

There have been several works relating to time based signatures in cryptography. In [HS90],

Haber and Stornetta present a method for time-stamping digital documents by sending a hash of them to a central server. The centralized server uses a hash chain to bound the time when a document was time-stamped at the central server, and the hash does not reveal the contents of a document to that server. In [M01], Mao introduces a scheme for time-released encryption and signatures. A large amount of non-parallizable computation is required before a special signature can be converted into a normal signature. However, this scheme is not correlated to absolute time (more computing power changes the time scale), and does not scale to allow a user to time-release multiple signatures simultaneously. A significantly more practical system was introduced by Mont, Harrison and Sadler in [MHS03]. Their system makes use of the IBE system of [BF01] to reveal signatures on documents (or the documents themselves) at a specified point of time in the future. Documents are encrypted with a public key for the date on which they are to be revealed. Each day, the central server reveals the IBE decryption key corresponding to that date.

While there have been many works on audit logs, we focus on those suitable for untrusted machines. In [SK98], Schneier and Kelsey propose a scheme in which log records are encrypted on the server, and an adversary who compromises the server is unable to read or alter existing records. The authentication of log entries is accomplished using a hash chain and a linearly evolving MAC key. To facilitate searching, records contain type information (analogous to extracted keywords in other schemes [WBD04, BCO03]), and the the owner of the log can allow a searcher to read selected records based on type information of those records. However, this scheme is not very efficient, as it requires one key to be sent from the owner to the searcher for *each* record to be searched, and the searcher must retrieve *all* records from the server to verify the hash chain. Furthermore, the type information of each record is stored in the clear.

The authors extend their scheme in [KS99] to reduce the number of keys the owner is required

to send to a searcher. Their solution involves a tree based scheme for deriving keys in which a non-leaf key can be used to derive all of the keys in its subtree. Thus, the owner can give one key to the searcher, who can then use that key to derive a range of keys. If the levels in the tree correspond to different time periods (i.e.: seconds, minutes, hours, etc.), their scheme can be used for time limited searching. Unfortunately, to keep the scheme secure and efficient, it only allows for a limited set of record types. This limitation is problematic for an audit log, where the extracted keywords may include, for example, an IP address. We extend their scheme by supporting more efficient searches, allowing richer type information (a set of extracted keywords) and present a more rigorous analysis of the number of keys required to be transferred to cover an interval of time in a tree based scheme. More details about these schemes are discussed in Section 4.2.

Chapter 3

Settings and Goals

To begin, we informally state our setting and goals. An entity, denoted as Alice, owns a server which she either operates directly, or has delegated operation to some third party. Alice is completely trusted and always protects her secret keys from compromise. To help justify this premise, she remains (somewhat) isolated; communication with her is infrequent and does not use a large amount of bandwidth. In practical terms, she may be considered to use a mobile device with limited storage capacity, processing power, and bandwidth. Unlike Alice, the server is not fully trusted and may be compromised by an adversary at some point in time, but until then, we assume that it diligently follows all protocols in the system. To provide evidence detailing an attack, the server maintains an audit log, which is generated by events occurring on the server (for example, alerts from an intrusion detection system). In the event that the server is compromised, before Alice can make use of the audit log, she must ensure that the log was not tampered with (e.g, determine if records were inserted, altered, or deleted). In order to allow her to remain isolated, she requires the ability to delegate searching and verifying selected parts of the log to third party investigators. Examples of investigators may be a managed security company hired by the owner, or a law en-

forcement agency presenting a search warrant to the owner. When referring to investigators, we use feminine pronouns.

Alice may restrict the scope of an investigator's search by time or keyword (each log entry has an associated set of keywords, for example, the IP address of a remote action), but for privacy reasons, it is important that an investigator does not learn anything outside of the scope of her investigation. All of the records on the server are encrypted, so the server can return any record that is requested without needing to perform authentication and without exposing private information. This setup permits an investigator to retrieve records directly from the server. Once the records are retrieved, Alice can verify their authenticity to the investigator (using very limited bandwidth), and the investigator can be satisfied of the completeness of a search. Furthermore, the investigator can perform the entire search and verification procedure efficiently.

We define the adversaries of the system in terms of investigators and the knowledge that they possess. When Alice allows an investigator to perform a search, she provides the investigator with one or more trapdoors to perform that search. While an honest investigator would delete these trapdoors once the search is complete, investigators are not trusted to always do so. In particular, for an investigator, we consider her knowledge of trapdoors from previous searches to be a vulnerability to the system. These trapdoors allow an investigator to retrieve certain parts of the log and verify those parts' authenticity and completeness.

As an additional threat, if an investigator "captures" the server, either by a remote exploit, or by physically taking the server, she may have more knowledge than before (e.g., access to the server's memory). Such an adversary may also be concerned with concealing its behavior. An explicit goal of the system facing such an adversary is to prevent her from searching the audit log and to prevent her from undetectable modifying the audit log. In systems where these goals are met, it becomes

significantly more challenging for such an adversary to obscure their activities.

The rest of this Chapter discusses our assumptions and our specific system requirements. Before we can present the requirements, we introduce some notation and terminology. The requirements are broken up into three groups. The first set of requirements outline how investigators receive trapdoors for searches, and what happens when they share them. The second, specifies the requirements and restrictions when an investigator performs a search. Finally, the third set of requirements address issues of restart recovery, in particular, what must be ensured to allow the server to resume normal operations after a compromise, without resorting to the trivial solution of starting a new log from scratch.

3.1 Assumptions

Our explicit assumptions are as follows:

Assumption 1 *The owner, Alice, is completely trusted. That is, she always fully protects her secrets, is not compromised during the lifetime of the system, and always follow the protocols of the system.*

Assumption 2 *All of the protocols and algorithms occur after Alice and the server are able to securely exchange some information establishing the existence of the log and share several keys. An example of a protocol to securely create a log is given in [SK98]. Secrets may be shared over a secure channel, or with a key agreement protocol.*

Assumption 3 *The server requires periodic interaction with Alice to receive the time and authentication keys required for a predetermined amount of time in the future, which we denote as I_{key}. We assume that these interactions are always successful and as such, keys are always available on the server when needed.*

The length of I_{key} can be varied based on the needs of the system. Less frequent contact between Alice and the server require more keys to be transferred at once, and thus exposed if the server is compromised. On the other hand, more frequent interaction makes the system more vulnerable to denial of service attacks.

3.2 Variables, Functions, and Notation

Subscript i's are used for normal log records, p's are used for time periods, and j's are used for each keyword associated with a record. In the audit log, there are two types of records. R_i's denotes normal log records (which encapsulate a message m_i) and A_p's denote anchor records. When referring to a generic log record, X is used. The function $time(X)$ returns the time period in which X was logged (it is assumed that no record is logged on the exact boundary between two time periods).

The function $words(X) = \{w_1, \ldots, w_j, \ldots, w_\ell\}$ returns the set of keywords that are relevant to the record X. Note that for normal log records, this function applies to the message embedded in the record, e.g., $words(R_i) = words(m_i)$. When referring to a specific record (e.g, R_i or A_p), we include the subscript of that record in each w_j. For example, the keywords associated with record R_i are $words(R_i) = \{w_{i1}, \ldots, w_{ij}, \ldots, w_{i\ell_i}\}$. Each keyword w_{ij} is used with some metadata stored on the server to create an index fragment, c_{ij}; more details about index fragments are in later chapters. Note that the $words(m)$ function must be deterministic, but the order that the keywords are returned in can be random.

The system uses several types of keys. K_i denotes the symmetric key used to encrypt m_i in each R_i, and K_w^{next} denotes the symmetric key to be used in the next record associated with w. The other two types of keys are dependent on a time period, p: TK_p denotes the time key and AK_p denotes

the authentication key.

We use both a, b and $a|b$ to denote concatenation. It is implied that each field has a well defined length, or a known delimiter to separate it from the next field. Different types of grouping symbols (parenthesis, brackets, angle brackets, etc.) are not significant; they are used for readability (so that the same grouping symbol does not appear nested inside itself).

3.3 Investigators and Keys

An investigator may have been authorized to perform several searches in the past, and may have kept the trapdoors obtained from all of those searches. Let $W = \{w_1, \ldots, w_v\}$ be the set of v keywords whose IBE trapdoors have been provided by the owner, and let $T = \{T_1, \ldots, T_d\}$ be the set of time periods for which time keys can be obtained from the trapdoors provided by the owner. More specifically, the time periods in T form d disjoint intervals (the T_i's) of average length t. Let $\mathcal{INV}(T, W)$-adversary denote an investigator with knowledge of T and W. The set of records that Alice allows such an investigator to read is defined as $match(T, W) = \{X : time(X) \in T \cap (words(X) \cap W \neq \emptyset)\}$.

The requirements related to investigators obtaining and sharing trapdoors are as follows:

Requirement 1 (Efficient Delegation) *Alice can authorize an investigator to search for records in* $match(T, W)$ *using bandwidth* $O(v + d\log(t))$.

Informally, a time interval is aligned if it spans an entire minute, hour, day, month or year. For example, the time interval of March 2004 is aligned. For some properly-aligned intervals of time, the $\log(t)$ factor becomes $O(1)$. In practice, we believe that most searches will be aligned.

Requirement 2 (Fine Granularity Searches) *Searches intervals may be begin or end on single second boundaries (or some other unit, as predetermined by Alice). A record may be described by any number of keywords, and each keyword may be of arbitrary length.*

The duration of the minimum interval of a search is called a *time zone*, which is discussed in more detail later. Servers that experience low volumes of traffic may wish to save resources by having larger time zones, for example, one per minute. More precisely-defined searches can be performed if there is more information contained in the keywords describing a record. At an extreme, satisfying this requirement means that an entire message may be extracted into its keywords.

For a particular search, an investigator is allowed to search for a keyword w if $w \in W$, and may not search for w otherwise. However, a new proposal by Sahai and Waters [SW04] may allow for Fuzzy Identity Based Encryption in which an investigator may search for w if it is "close" to any element of W.

Requirement 3 (Strong Collusion Freeness) *The set of records that can be read by a group of k adversaries, $\mathcal{INV}(T_1, W_1)$, ..., $\mathcal{INV}(T_k, W_k)$ who share their trapdoors (e.g., the records in $match\left(\cup_{i=1}^{k} T_i, \cup_{i=1}^{k} W_i\right)$) is equivalent to the records that can be read by these investigators sharing their results (e.g., $\cup_{i=1}^{k} match(T_i, W_i)$).*

This requirement formalizes the intuitive notation that investigators cannot learn any more records by combining their trapdoors. Unfortunately, our scheme cannot achieve strong collusion freeness due to the independence of per-keyword trapdoors and per-time period trapdoors, as well as the homomorphic nature[4] of $match$. In particular, when $i \neq j$, the records of $match(T_i, W_j)$ can be read by investigators who combine their trapdoors, but not by any individual investigator.

[4] That is, $match(T_1 \cup T_2, W) = match(T_1, W) \cup match(T_2, W)$ and $match(T, W_1 \cup W_2) = match(T, W_1) \cup match(T, W_2)$.

In order to achieve Strong Collusion Freeness, there need to be separate sets of time keys for each keyword; however, maintaining these separate sets presents a problem for the requirements of Efficient Delegation and Fine Granularity Searches. For example, the scheme of [KS99] needs to severely limit the number of possible keywords to remain efficient. Moreover, when only a handful of keywords exist, all searches become very general in scope. Furthermore, [KS99] has another significant limitation: the keywords for each record are stored in the clear.

Thus, we introduce a weaker notion of collusion freeness which balances the other requirements related to efficiency and flexibility, but allows investigators to perform searches that Alice may not have intended to allow. Future work will examine how to achieve Strong Collusion Freeness without limiting other requirements. More details about these tradeoffs are discussed in Chapter 5.

Requirement 4 (Weak Collusion Freeness) *The set of records that can be read by a group of k adversaries, $\mathcal{INV}(T_1, W_1)$, …, $\mathcal{INV}(T_k, W_k)$ who share their trapdoors (e.g., the records in $match\left(\cup_{i=1}^{k} T_i, \cup_{i=1}^{k} W_i\right)$) is equivalent to the records that can be read by a $\mathcal{INV}(T, W)$-adversary (e.g., $match(T, W)$), where $T = \cup_{i=1}^{k} T_i$ and $W = \cup_{i=1}^{k} W_i$.*

In other words, a group of investigators who share their trapdoors cannot learn more than a single investigator who has all of those trapdoors, or put more simply, keyword and time period trapdoors are not investigator specific. Consequently, more records may be obtained when investigators combine their trapdoors rather than combine their individual search results. This requirement is achieved as a result of the homomorphic nature of $match$.

However, a positive side effect of the weaker requirement is that Alice does not need to keep state about the prior trapdoors given to each individual investigator. To determine what any investigator could possibly read, she simply assumes that all investigators collude, and only needs to keep

track of which trapdoors have ever been provided to any investigators. Although the owner does not need to keep track of all previous searches to use the system, we suspect that it is prudent to do so.

3.4 Investigators and Searches

Once an investigator has obtained a set of trapdoors T and W, either from Alice or through collusion with other investigators, she can perform a search. This section discusses the requirements of those searches.

Requirement 5 (Completeness) *The investigator can read all log records $X \in match(T, W)$, and be convinced that no other records in this set exist in the log. The investigator can authenticate all of these records without contacting Alice.*

The authentication is performed by verifying one public key signature for each of the d time intervals being searched. The public key for that signature was previously provided by Alice during the search delegation. Thus, Alice may remain off-line once the investigator obtains the trapdoors needed to perform the search.

Requirement 6 (Efficient Searching) *For each keyword $w \in W$ and each interval of time $T_i \in T$, once the first matching record is found, an investigator will only need to retrieve $O(k)$ records, where k is the total number of records containing w during T_i, e.g., $|match(\{w\}, T_i)|$.*

A consequence of efficient searching is that the total number of records retrieved (excluding those before the first match is found), is at most a constant factor (in our case, 2) above the number of records that actually match. Other schemes previously proposed (in particular, [SK98, KS99, WBD04]) require that all of the records in the log (or for a period of time in the log) be retrieved.

In our case, we only need to retrieve a number of records proportional to the number of matching records. Thus, when searching for infrequently occurring keywords, our scheme is much more efficient than previous work. While there may be many records retrieved before the first match is found, our performance until the first match is equal to that of [WBD04] when their indexing optimization is used.

Note that we are not concerned with the sizes of records stored on the server. Since an investigator only needs to retrieve a minimal number of non-matching records, it is acceptable for these records to be larger. Moreover, storage is cheap, whereas time and bandwidth are scare resources.

Requirement 7 (Privacy Preserving Searches) *While performing a search, an investigator cannot read log records* $X \notin match(T, W)$.

Requirement 7 warrants further discussion. To allow an investigator to be convinced that she has obtained all records related to w for a search of time interval T_i, she must be convinced that the no other matching records related to w occurred after the start of T_i. However, retrieving and examining all other records during T_i is inefficient. Moreover, she must not be allowed to learn the exact time period in which the previous record related to w prior to T_i was logged, nor can she learn if any such record exists.

3.5 Recovery from Compromise

After a server has been compromised, it may be difficult to determine if an adversary installed any backdoors that allow for future access. As a result, many servers are simply formatted and their software reinstalled after a compromise. Part of that process may typically include starting a new audit log. While this simple solution is sufficient, it has limitations and overhead which become

clear when one wants to perform searches spanning both the old and new audit logs, especially when the server may be frequently compromised and reinstalled.

In our system, the audit log is tamper resistant (as defined below), and as such, it is possible to determine which part of the log is intact. The untrusted remainder of the log can be removed, and our system allows the most recent good record to be "spliced" to a new log segment, and then allows the log to continue normal operation. The advantage of our approach is clear – a single set of trapdoors can allow an investigator to perform a search spanning multiple segments of the log.

Define a $\mathcal{CAP}(T, W, p)$-adversary as a $\mathcal{INV}(T, W)$-adversary who compromises the server during time period p. If this adversary was not a normal investigator before capturing the server, or is not colluding with an investigator, then the sets T and W are empty. Such an adversary has access to everything stored in memory on the server, and has read/write access to everything on the server's stable storage, including the current time key, TK_p. This type of adversary is in control of the server, and can clearly prevent any future log records from being written. In addition, log records for the current time zone are vulnerable to being undetectable deleted, motivating a tradeoff between overhead and security in the choice of a time zone length.

The goal for a $\mathcal{CAP}(T, W, p)$-adversary is to be *certain* that the log contains no record of their intrusion. If the system prevents her from searching the log, then she cannot be certain that an intrusion detection system on the server did not log her intrusion. Thus, the adversary is forced to assume that an incriminating log record exists, and she must alter the log to remove the possibility of being caught. If the system can also prevent the adversary from undetectable altering the existing log records from before time period p, then it is impossible for the adversary to achieve her goal. As a result, the specific requirement for the system to allow normal operations to resume after a $\mathcal{CAP}(T, W, p)$-adversary are:

Requirement 8 (Tamper Resistant Log) *The adversary must not be able to insert, delete, or alter any log records from any time zone before p without such changes being detectable by Alice or another investigator that Alice delegates such a verification to.*

Requirement 9 (Limited Searching) *The adversary cannot read any more records than an $\mathcal{INV}(T \cup p, W)$-adversary.*

A side effect of having the abilities of an investigator with knowledge of TK_p is that the adversary can perform a search for keywords in W during the current time period (p). However, since the current time period's authentication key, AK_p is stored on the server, the adversary can simply delete all records for the current time zone without leaving a trace. Thus, we reiterate the need to carefully chose the length of a time zone. For an interactive adversary (e.g., a hacker manually typing in commands), a time zone length of one minute may be sufficient, but for an automated system, the time zone may need to be reduced to a single second.

Requirement 10 (Limited Key Exposure) *All future keys (that is, those for time periods after I_{key}) cannot be derived from the secrets that are stored on the server during time period p.*

As a result of this requirement, after the compromise is detected, the system is cleaned, and a finite number of time periods have elapsed after p, the server can resume normal operations. Recall the time period length is related to Assumption 3.

Chapter 4

Preliminaries

This chapter will introduce the cryptographic primitives used in the rest of the thesis and provides a more detailed description of the related work that our system directly builds on.

4.1 Cryptographic Primitives

A cryptographic hash function is an efficiently computable mapping from variable length messages to fixed length digests, and is denoted as $h(m) = d$. Hash functions are collision resistant (difficult to find an x, y such that $h(x) = h(y)$), and are one-way (difficult to find an x such that $h(x) = d$ for some fixed d). In our system, we use SHA-1 [F180], which produces a 20 byte (160 bit) output. When more than 160 bits are needed as the output of a hash function, we extend SHA-1 by outputting $\langle h(m)|h(m+1)|h(m+2)|\ldots\rangle$.

A message authentication code (MAC) is a hash function that uses a secret key k, and is denoted as $f_k(m) = d$. If k is known, $f_k(m)$ can be computed efficiently. However, if k is not known, an adversary cannot compute digests for messages nor determine if a digest matches a particular

message. An example of a MAC is HMAC [F198].

A symmetric key cipher is a deterministic pseudorandom permutation on fixed size blocks, E, that uses a secret key k. Encryption of a message is denoted by $E_k(m) = c$, and decryption is denoted as $D_k(c) = m$. In our system, we use AES [F197] with both a 128 bit key and block size.

A public key signature scheme provides non-reputable signatures of messages. In practice, for efficiency, one typically signs a hash of the message instead. A secret signing key is used to generate signatures, and a public verifying key allows anyone to verify a signature. In our system, we use the Digital Signature Algorithm (DSA); for more details about this scheme, we refer the reader to [F186].

4.2 Tamper Resistant Audit Logs

In Chapter 2, we introduced a tamper resistant audit log scheme [SK98] and an improved version [KS99]. The goal of a tamper resistant audit log is that an adversary who has compromised the server cannot read or undetectable alter existing records. We first introduce the notation and terminology used in these schemes[5]. Message m_i is the ith message to be stored in the audit log. A set of permissions associated with m_i, i.e. $words(m_i)$, describe the contents of the message in enough detail for the owner of the log to be able to decide if a particular investigator should be allowed to read m_i, but with insufficient information to to reveal m_i. These permissions can be thought of as keywords extracted from m_i. An authentication key, AK_i, is stored by the server, and is evolved after each log record is written.

To store a new record, an encryption key $K_i = h(words(m_i), AK_i)$ is derived from the permis-

[5] For ease of exposition, we changed the notation from these papers to the notation used in our system.

sions associated with the message and the current authentication key. The resulting encryption of the message is $E_{K_i}(m_i)$. Next, a hash of this record and the hash from the previous record is created as $Y_i = h(Y_{i-1}, E_{K_i}(m_i), words(m_i))$. The hash from the previous record is included to form a hash chain. Finally, a MAC is used to authenticate the hash, $Z_i = f_{AK_i}(Y_i)$, and the authentication key is evolved ($AK_{i+1} = h(AK_i)$). The i-th log record is $R_i = (words(m_i), E_{K_i}(m_i), Y_i, Z_i)$. Once the record is written, AK_i and K_i are deleted.

We note however that there are two major problems with this scheme. First, the extracted keywords $words(m_i)$ are stored in the clear, which may leak some information to an adversary. As a consequence, there is an implicit limitation on how detailed the keywords may be, limiting the granularity of searches. Second, and more importantly, if an adversary obtains AK_p, she is able to derive all future authentication keys. This necessitates starting a new audit log after a compromise.

Note also that from time to time, an investigator may want verify or read a range of records in the audit logs, from R_b to R_f ($b \leq f$). She can accomplish this by retrieving *all* of the records between R_b and R_f from the server and verifying the hash chain (the Y_i's). Next, she sends the owner of the log f, Y_f, and Z_f, along with a list of $i, words(m_i)$ for *each* R_i (with $i \in [b, f]$) she desires to read. Then, the owner can verify the final MAC (Z_f) and provide K_i for each $words(m_i)$ that the owner determines the investigator should be allowed to read. A feature of this scheme is that if the investigator lies about $words(m_i)$, the corresponding key that the owner retrieves will not be correct. However, the number of messages that need to be exchanged to allow an investigator to perform a search is a significant disadvantage of this scheme, which was addressed by the authors in [KS99].

To reduce the number of keys that need to be transferred, in [KS99] the authors allow investigators to derive multiple encryption keys from a single secret. This construction is based on the use of

trees: a key for a node in the middle of the tree can be used to derive the keys for each of its children; iteratively executing this procedure will result in deriving all the keys for a subtree. The levels of the tree can be used to represent different lengths of time (e.g., hours, days, etc.). Thus, the owner of the log can allow an investigator to search a large portion of the log by providing a relatively small number of keys. Although the authors do not provide a rigorous analysis of the number of keys required, intuition suggests that it would be logarithmic in the duration of the interval.

Unfortunately, a consequence of the key reduction of [KS99] is that record encryption keys are no longer derived from permission masks, so investigators can lie to the owner about permissions. Thus, a separate tree of keys must be maintained for each possible permission mask. As a result, for the scheme to remain efficient (in terms of the number of keys that must be managed), the number of possible extracted keywords must be reduced to a very small number. This is of course problematic when information relevant to an audit log (for example, an IP address) are extracted as keywords; the ability to have millions of different possible keywords without creating a burden of key management becomes paramount (see *Requirement 2*).

4.3 IBE-Enabled Audit Logs

Our approach extends the asymmetric key scheme presented in [WBD04]. Their approach depends on Identity Based Encryption (IBE), which allows arbitrary strings to be used as public keys. We use the IBE scheme of Boneh and Franklin presented in [BF01] that achieves chosen ciphertext security. We denote an identity based encryption of a message m with public key w as $c = IBE_w(m)$, the private key corresponding to w as d_w, and the identity based decryption of c with private key d_w as $m = IBD_{d_w}(c)$. While the other cryptographic primitives described are relatively fast, identity based operations are very slow, and care must be taken to minimize the number

of IBE operations required.

At a high level, the scheme of [WBD04] is similar to the tamper resistant audit log schemes discussed in Section 4.2. In order to add a message m_i to the log, a random encryption key, K_i, is chosen, and m_i is encrypted with this key. The message has a set of associated keywords, $words(m_i) = \{w_{i1}, \ldots, w_{ij}, \ldots, w_{i\ell_i}\}$. K_i is appended to a public constant $flag$, and the result is identity-based encrypted using each $w_{ij} \in words(m_i)$ as a public key. The log record R_i contains $E_{K_i}(m_i), c_{i1}, \ldots, c_{ij}, \ldots, c_{i\ell_i}$, where each $c_{ij} = IBE_{w_{ij}}(flag|K_i)$. In addition, R_i contains a part of a hash chain and an MAC of that hash (similar to the Y_i's and Z_i's from [SK98]). Once the log record is written, K_i is deleted.

A major advantage of this scheme over that of [SK98] is that the keywords associated with each message are not stored in the clear. Moreover, an adversary who examines the ciphertext $c_{ij} = IBE_{w_{ij}}(flag|K_i)$ cannot determine which string was used as public key to encrypt it (e.g., w_{ij}). This result is due to the key-privacy property the IBE scheme of [BF01]; we refer the reader to [BBD01] for more information about key-privacy, and to [BCO03] for a proof of key privacy in IBE.

To perform a search for all records in the log containing the keyword w, an investigator requests the trapdoor (private key) for that keyword (d_w) from the owner of the log. Next, the investigator retrieves each record R_i from the log and attempts to decrypt each c_{ij} using d_w. She can determine if a decryption was successful by checking if $IBD_{d_w}(c_{ij})$ begins with $flag$. If the decryption was successful, she has obtained K_i, which can be used to decrypt m_i. We note that the use of the IBE scheme that provides chosen ciphertext security, rather than using the semantically secure version, makes the use of $flag$ unnecessary, as decryptions with the incorrect key will fail. However, to be consistent, we still describe their scheme with the use of $flag$.

Clearly, this scheme is has some limitations. An investigator must retrieve *all* records in the log, and attempt one IBE operation for each keyword associated with each record. Furthermore, IBE operations are much more computationally expensive than symmetric key operations. More importantly, knowledge of d_w will allow an investigator to decrypt *all* records R_i with $w \in words(m_i)$, even those that have not yet been created. As such, it is desirable for a scheme to limit the scope of an investigator's query to a specific period of time.

To address the performance reduction associated with the use of IBE, the paper also discusses some optimization techniques. One optimization, pairing reuse, allows an expensive operation to be cached and reused in future encryptions, since it only depends on the public key (and not the message). However, the most important technique presented is indexing, which allows an investigator to retrieve fewer records and perform fewer IBE decryptions during the course of a search. This improvement groups a small number of consecutive log records into a "block". Note that the block size does not need to be constant, it may be the number of records that appeared during a particular interval of time. Once all of the records in the block are stored (as $i, E_{K_i}(m_i)$), a special index record is written (analogous to an anchor record in our scheme). For each keyword w associated with any record in the block, the index record contains an IBE with public key w of a list of which log record $(i_1, \ldots, i_\ell)$ from the block contains the keyword w, and the encryption key used for each of those records $(K_{i_1}, \ldots, K_{i_\ell})$. See Figure 4.1 for an overview. While indexing provides a large performance increase, it can allow an adversary who compromises the server to read record in the current block. This vulnerability is a direct consequence of the server needing to keep each K_i in the clear until the index record is written.

Next, we consider the performance of indexing. Let n denote the number of entries in the log, b denote the number of log entries per block, u denote the average number of distinct keywords in

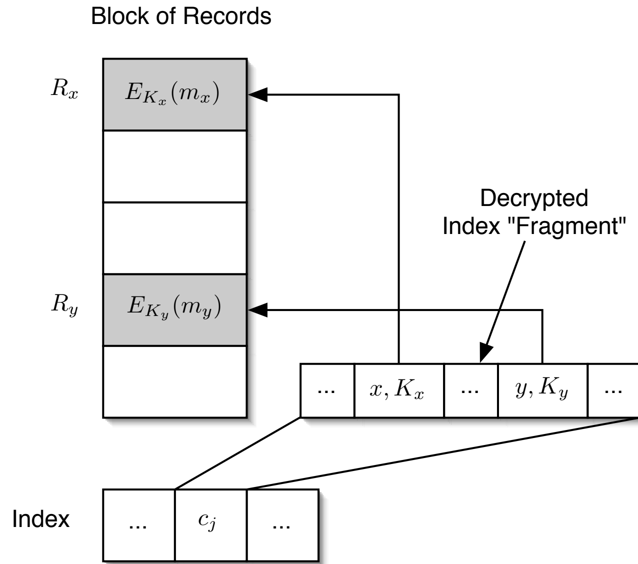

Figure 4.1: The indexing optimization from [WBD04]. Access to an index fragment in an index record allows an investigator to locate and decrypt all matching records within that block.

a block, and k denote the number of log records that are related to a keyword w. Then to perform a search for w, an investigator needs to retrieve $O(n/b + k)$ log records and perform $O(u \cdot \frac{n}{b})$ IBE decryptions. In practice, we expect that most searches would be performed on infrequently occurring keywords, and as such, $n/b \gg k$. A consequence of keywords occurring infrequently is that there are many distinct keywords in each block, so $u > b$. Thus, the number of records to retrieve and the number of IBE operations to perform are proportional to the total number of records (n), rather than the optimal value (the number of matching records, k). By contrast, the number of records retrieved and IBE decryptions performed during a search are both $O(k)$ in our construction, once the first matching record is found.

Chapter 5

Time Zones

This chapter discusses several possible approaches of adding support for time-scoped searches to the scheme of [WBD04] (see Section 4.3). Through a discussion of the limitations of each of these schemes, we motivate the system design choices and requirements presented in Chapter 3. We show that in order to efficiently support a large number of keywords in a per-keyword time zone setting, the master IBE secret would need to be stored on the server. Since any adversary who compromises the server would learn this master secret, it is not practical to have separate time zones for each keyword. Instead, all keywords are part of the same time zone (e.g., only weak collusion resistance is provided). In this setting, the server only requires periodic interaction of constant size with the owner to update its key material for a predetermined number of time periods in the immediate future (I_{key}). Furthermore, we show that the extra encryption used with a time based key should encrypt the c_{ij}'s so that less information is available to an adversary. Since the most efficient solution involves only a single time zone, then this necessitates that log records are themselves encrypted.

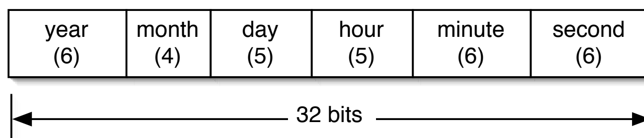

Figure 5.1: Time periods are 32-bit unsigned integers. Each of the parts of the date is broken into fixed length fields, with the lengths as indicated. This representation allows all dates between 1970 and 2033 with one second precision to be used.

5.1 Time Periods

We begin by illustrating how 32-bit unsigned integers are used to represent time periods. Rather than using a standard Unix timestamp (the number of seconds since January 1, 1970), we split the 32-bits into several fixed-width fields (see Figure 5.1). As a result, we do not have the ability to represent extended periods that Unix timestamps can; however, our approach still allows representation up to 2033 with 1-second precision. In practice, we expect most search requests to be aligned to some period of time (for example, days), and in such cases, we believe that our representation allows for a more efficient implementation.

A consequence of having separate fields for each part of a date is that some 32-bit values represent invalid time periods (e.g., the 15th month of the year, or the 30th hour of a day). These invalid integers cannot be used as time periods, but internally, if a field is set to its minimal valid value, it is interpreted as all 0 bits. Similarly, if a field is set to its maximum valid value, it is interpreted to as all 1 bits. This means that the year, hour, minute, and second fields are 0-based; the day and month fields are 1-based.

A *time zone* is the finest granularity with which time scoped searches can performed, and consists of a continuous interval of time periods. The duration of a time zone should be chosen such that there are several log records added to the log during each time zone. Time zones can be a single second, but in practice, we expect them to be aligned on minute boundaries. When we refer to a

time period p, we mean the last time period occurring in that time zone, and by $p-1$ we mean the last time period occurring in the previous time zone.

To simplify the analysis, we assume that there are roughly the same number of log records per time zone. While it is true that most servers experience spikes in usage, we believe that these constant factors are captured by the big-O notation. Recall that n is the number of entries in the log, and b is the number of records per time zone (e.g., in a single block). Thus, the number of time zones in the log is n/b.

5.2 A Trivial Solution

The trivial way to add time-scoped searching to [WBD04] is as follows. First, the time period that a record R_i was added during, $time(R_i)$ is stored in that record. Note that the time period can be stored in the clear since the time period of the record can be inferred from its relative position in the log. Next, the public key used to form c_{ij} is changed from w_{ij} to $time(R_i)|w_{ij}$. Thus, $c_{ij} = IBE_{time(R_i)|w_{ij}}(flag|K_i)$. To use the indexing enhancement, one needs to ensure that all of the records in a block occur during the same time zone.

Searches are performed as in the original scheme. An investigator asks Alice for the trapdoors corresponding to each keyword she wants to search for during an interval of time. Unfortunately, this simplistic scheme results in a large performance penalty for Alice. To search for a single keyword in t time zones, she needs to provide an investigator with $O(t)$ IBE keys. However, from the investigator's perspective, performance is comparable to before. The number of index records to retrieve and the number of decryptions required is proportional to the fraction of log the within the scope of the search, $\frac{t}{n/b}$.

Although this trivial approach places a great burden on the owner, it has a desirable feature that

the time zones are separate for each keyword. This interdependence between key types allows Alice to prevent the problem of unintended search capabilities discussed in Section 3.3, and allows Strong Collusion Freeness (see *Requirement 3*) to be achieved. However, efficient search delegation is an important goal (see *Requirement 1*), so an alternate approach is needed.

5.3 A Tree Based Approach

To reduce the number of keys that the owner needs to provide an investigator, we use a tree-based technique to derive time keys similar to that of [KS99]. Note that this technique is also used to derive authentication keys, but for ease of exposition, we limit the discussion to the use of time keys. A tree based scheme satisfies *Requirement 10*: given a time key, all future time keys cannot be derived. A linear key evolving scheme (for example, a hash chain) cannot satisfy this requirement since knowledge of one key can be used to derive *all* future keys.

Recall that TK_p is the time key for time zone containing time period p. We extend this definition to allow for time keys to be generated in a tree structure by having "partial" time periods: 32-bit numbers with some of their most significant bits determined, but with their least significant bits not yet determined. We denote by $|p|$ as the bit length of p; $low(p)$ as the smallest 32-bit unsigned integer that can be obtained from p (e.g., p followed by all 0's); and $high(p)$ as the largest 32-bit unsigned integer that can be obtained from p (e.g., p followed by all 1's).

A *master time key*, which we denote as TK_*, is randomly chosen to be at the root of the tree (implicitly, TK_* has $|p| = 0$). Let $x \in \{0, 1\}$ denote a single bit, and $h_x(m)$ be a pair of hash functions that are publicly computable. There are several possible constructions, for example, a MAC $f_x(m)$ or a concatenation $h(x|m)$. Then we define $TK_{p|x} = h_x(TK_p)$, provided that $|p| \in [0, 31]$. Once $|p| = 32$, TK_p is an actual time key. As an example, if time periods were one month

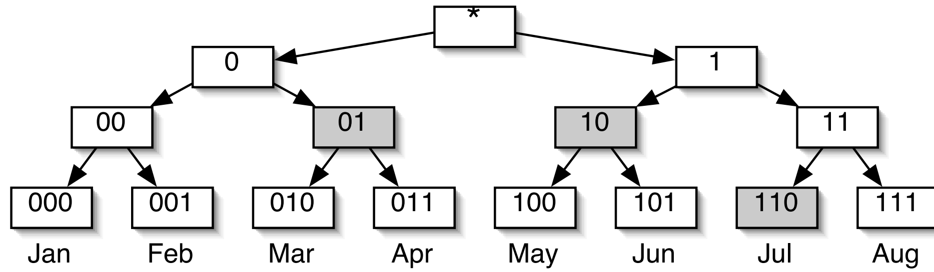

Figure 5.2: Consider a simple example where time periods are one month in duration, and the search interval spans March through July. The values in each box are the subscripts of time keys. To allow an investigator to derive the keys for this search interval, the keys in the shaded boxes need to be transmitted to her.

in length, then the minimum number of keys that must be transmitted to the investigator for the period spanning March through July is depicted in Figure 5.2.

Using a tree based scheme, on average, $O(log(t))$ keys need to be sent by Alice to an investigator for a search spanning t time zones. Furthermore, if the interval of time to be searched is well aligned (e.g., its boundaries correspond to one unit of time, for example, one month), $O(1)$ keys can be provided to an investigator to generate keys for the search. Since the time keys are independent of the keyword trapdoors, this tree based construction meets the requirements of Efficient Search Delegation (*Requirement 1*).

Now that Alice has a mechanism to efficiently distribute time keys to an investigator, we address the question of how to integrate these keys into the audit log on the server. The server is required to have some key material (for a limited interval of time surrounding the present, I_{key}, as discussed in *Requirement 10*), and requires periodic contact with Alice to obtain keys for future periods. While it is true that more key material is now stored on the server, our tree based construction limits the exposure of future keys when the server is compromised. This is a direct contrast to the schemes of [KS99, WBD04], where all future versions of the linearly derived authentication key are exposed when the server is compromised.

Clearly these keys cannot be prepended to keywords and used as IBE public keys (which is the case for the trivial scheme) since the resulting construction requires Alice to distribute as many keys for searches as in the trivial scheme. Thus, we use the time based keys as symmetric keys to encrypt some or all of each log record. The two possible ways of encrypting a record are to encrypt the entire record, or to encrypt individual c_{ij}'s in the record. These options are explored in the next two sections.

5.4 Per Keyword Time Zones

First, we consider schemes that have separate time zones on a per-keyword basis. Recall that we want to have as large a number of keywords possible (see *Requirement 2*), and cannot require the server to store a master key (see *Requirement 10*). Let $TK_{w,p}$ denote the time key that corresponds to keyword w during time period p. Recall that $c_{ij} = IBE_{w_{ij}}(K_i)$; the $flag$ is not required with the chosen ciphertext secure version of IBE. As previously discussed, we cannot use a value other than w_{ij} for the public IBE key. Thus, the design choices left to explore are either to encrypt K_i or c_{ij}.

Let $c'_{ij} = E_{TK_{w_{ij},p}}(flag|c_{ij})$ denote the encryption of c_{ij}. Recall that the time period, $p = time(R_i)$ during which record R_i was recorded is stored in the clear. Thus, when an investigator is searching for records associated with a keyword w, and the time period p is within the scope of her investigation, each c'_{ij} in a record R_i can be decrypted with $TK_{w,p}$. If the keyword for c_{ij} was actually w, then the decryption $D_{TK_{w,p}}(c'_{ij})$ will be of the form $flag|c_{ij}$. Upon observing $flag$, she can perform an IBE decryption on c_{ij} to obtain K_i. On the other hand, if $w_{ij} \neq w$, then $D_{TK_{w,p}}(c'_{ij})$ will not begin with $flag$, and she does not need to perform an IBE decryption. Since symmetric key operations are much more efficient then IBE decryptions, searches will actually be faster, since

unnecessary IBE decryptions can be avoided.

By contrast, if K_i is encrypted first, and $c''_{ij} = IBE_{w_{ij}}(E_{TK_{w_{ij},p}}(K_i))$, an investigator will need to IBE decrypt each c''_{ij} in R_i. Thus, the performance is the same as for the original scheme. Hence, it make more sense to encrypt c_{ij} with a symmetric key scheme (after the IBE encryption) rather than encrypt K_i before the IBE encryption. We emphasize that to obtain significant gains in searching efficiency, unnecessary IBE decryptions must be avoided. In particular, the optimal number of IBE decryptions to perform is one per matching record, which intuitively requires some type of index structure.

In order for the server to be able to encrypt c_{ij} and form c'_{ij}, it requires $TK_{w_{ij},p}$. In the tree based scheme we described, efficient protocols exist for deriving keys in different time periods (p's), but not for different keywords (w_{ij}'s). In particular, we need a protocol that allows the server to obtain key material for some w_{ij} that it has not previously seen. This problem is explained by Kelsey and Scheiner in [KS99]:

> Our improved scheme requires the ability for [the investigator] to compute keys for himself. This means we cannot use our permission mask mechanism any longer. Instead, we must use a much less efficient mechanism. We will have to maintain a whole set of keys for deriving record keys, as described above, for each different permission mask. This means that instead of potentially having millions of different permission masks, we will likely end up with two or three.

Recall that their notion of permission masks are analogous to extracted keywords. It is essential to support a large number of possible keywords, especially considering an audit log could have keywords corresponding to IP addresses (see *Requirement 2*). Thus, it is not acceptable to only have two or three possible keywords. If separate time key trees are used for each keyword, the periodic key updates from Alice will require bandwidth linear in the cumulative number of distinct keywords in the log. This is far in excess of the constant bandwidth required when only one tree of

time keys is used.

In order for the server to be able to generate keys for arbitrary keywords without contacting Alice, it must store a "master" secret, which we denote by $\overline{TK}$, used to generate the master time key $TK_{w,*}$ for each keyword w's time key tree. However, in the event that the server gets compromised, an adversary would be able to remove all time zone restrictions in the system – they could simply derive all time keys for all keywords on their own from $\overline{TK}$. This vulnerability motivates exploring the use of time zones that are not based on keywords.

5.5 A Single Time Zone

The main advantages of using one time zone is that the server does not need to store a master secret, ($\overline{TK}$ or TK_*) and can support a large number of distinct keywords (*Requirement 2*). Before we discuss how to implement a single time zone, we need to reiterate the problem with a scheme in this setting: an investigator who has obtained trapdoors from multiple searches has the ability to combine those trapdoors and read records that were not readable during any individual search (see *Requirement 4, Weak Collusion Freeness*).

An example of this problem is as follows. If Alice allows an investigator to search for w in January, and later allows her to search for w' in February, she will also be able to search for w' in January and w in February. Alice did not intend for records matching either of these two conditions to be read by the investigator. Moreover, Alice cannot know what records an investigator is actually able to read after being provided with trapdoors for another search, as the investigator may have colluded with other investigators and obtained additional trapdoors. However, this means that Alice is not required to keep state for the trapdoors that have been provided to each individual investigator. A limitation of our construction is that we can only provide Weak Collusion Freeness. In future

work, we hope to modify our construction to provide Strong Collusion Freeness.

Recall that $R_i = (i, time(R_i), E_{K_i}(m_i), c_{i1}, \ldots, c_{i\ell_i})$. For clarity, let $p = time(R_i)$. To integrate time limited searches into the log, the time key for TK_p can be used to encrypt a part of R_i, or R_i in its entirety. For the remainder of this section, we consider each of these cases, and discuss the tradeoffs between them.

The message m_i is encrypted with a random key K_i. Let $m'_i = E_{K_i}(m_i)$. Instead of storing m'_i in R_i, we can replace it with $E_{TK_p}(m'_i)$. The cost of implementing this change on the server is one additional symmetric key encryption, which is negligible when compared to the IBE operations needed to produce the c_{ij}'s. Since the time period p is stored in the clear, an investigator does not need to perform any IBE decryptions on the c_{ij}'s if p is not in the time interval she is searching. Otherwise, the investigator can proceed as before, and if K_i is obtained from one of the c_{ij}'s, one additional symmetric key decryption is needed to decrypt m'_i and obtain m_i. Note that the costs are the same if m_i is encrypted with TK_p before K_i.

An alternate approach is to encrypt m_i with a key other than K_i. The new key, which we denote by K'_i, is derived from the original key and the time key, for example, $K'_i = h(K_i, TK_p)$. This approach is almost identical to the previous version, except that an extra hash is required rather than an extra symmetric key encryption. Since both a hash and a symmetric key encryption are much quicker than an IBE encryption, it does not matter which is used.

The final part of the record that can be encrypted are the c_{ij}'s. Since the time key is independent of the keyword, it makes sense to encrypt all of the c_{ij}'s as a single block. This amounts to encrypting all of R_i, except for i, p, and $m'_i = E_{K_i}(m_i)$. Since i and p can be inferred from a record's position in the log, and we previously discussed the double encryption of m_i, we only need to discuss encrypting the entire record. As before, the server only needs to perform an extra symmetric

key encryption. However, when an investigator is performing a search, she may not know the *exact* time period of an arbitrary record. As a result, a large number of symmetric key decryptions and extra record retrievals are required to find the first record of the time interval being searched. Note that once the time period of one record is known, only one or two extra decryption is required per record, since the records are stored in order – the following record is from the same time period or the next one (the cost of gaps can be amortized over the entire search).

While encrypting part of a record (e.g., m_i') appears to be more efficient, it turns out that encrypting the entire record provides much a better construction to build on. In the next two chapters, we show how we can encrypt the entire record and avoid the inefficiencies the investigator has with searching. In particular, preventing access to the c_{ij}'s can be used to build an index and provide time zones.

Chapter 6

An Index for Efficient Searching

As previously discussed, there are significant limitations and tradeoffs that need be considered when designing schemes that achieve time limited searches. We address these issues by having efficiency be our primary goal. In the next chapter, we integrate the ideas of time limited searches and the use of an index to achieve our goals.

The general idea of efficient searching involves an index. However, when designing an index, one needs to keep in mind that an important goal is to ensure that the untrusted server should not need to store a large amount of key material, and the material should be evolved to achieve forward security. Moreover, the index needs to be secured, as it presents a valuable target for an adversary. Hence, the index needs to be designed very carefully.

For an index to be useful, an investigator must be able to retrieve records from the server with random access, rather than only be permitted sequential access. At a minimum, each record has an sequence number (its ordering in the entire audit log), which for log record R_i is simply i. Log records can be efficiently looked up by the server using the sequence number. Other types of efficient lookups are possible, for example, by a hash of R_i.

Without an index, completeness is guaranteed by examining every record in the log. Moreover, the hash chain use for integrity ([KS99, WBD04]) involves all records in the log. An index allows an investigator to perform a search while only examining a subset of records in the log. Thus, the index must be able to preserve the guarantees of completeness and integrity that exist.

6.1 A Separate Index

The simplest form of an index is a linear structure that is separate from the log. Specifically, for each keyword w, there is an index L_w that is an array of sequence numbers of records that contain w. Note that storing the actual K_i's for those records, rather then just their sequence numbers, would allow for more efficient searches, but would make the index even more valuable (considering the index is stored in the clear). In this setting, to search for a keyword w, only the records with their sequence numbers in L_w need to be searched. As a result, the number of IBE decryptions needed is reduced to one per keyword per record in the index from one per keyword per record in the entire log.

However, this simple solution has a large drawback. In order to add a new record, the server needs to identify which index is which, thus, the w of L_w is in the clear. Furthermore, each sequence number within L_w is in the clear, so the server has knowledge of which records contain which keywords. From an adversary's perspective, such an index can be easily abused, and a less revealing solution is needed.

6.2 An Integrated Index

To protect the index from an adversary, the only time an index should be accessible is, from some log record R_x that uses keyword w, when an investigator wants to find the next log record (say R_y, for some $y > x$) that uses keyword w. Since we are in an append only model, when the record R_y is added, R_x was already written and cannot be altered to point to R_x. As a result, the integrated index will be in the form of “back pointers” (which we also call reverse pointers) stored as part of each c_j. This construction implies that searches are now started at the end of the log (the present), and progressively work there way toward the beginning of the log.

In the original scheme of [WBD04], each keyword w_j in record R_y is used to encrypt $c_j = IBE_{w_j}(flag|K_y)$. Now, we want to include the sequence number of the previous record containing keyword w_j (e.g., x from R_x). Thus, we now have $c_j = IBE_{w_j}(flag|K_y|x)$. When an investigator is searching for a keyword w, she starts at the end of the audit log (e.g., the most recent entry). Next, she attempts to decrypt each c_j for that record using the private key for w. If the record does not use the keyword w, no decryption is successful, and the investigator proceeds to the previous record and repeats this process until the first match or the beginning of the log is reached.

Note that an additional optimization is possible with an index. In addition to the sequence number, x, of the previous record containing a keyword, c_j can also contain which of the c_j's in the previous record (R_x) was for the keyword w. We denote by c_{j^*} the correct c_j from the previous record, and include j^* in c_j. Using this optimization, an investigator only require one IBE decryption per matching record, rather then an IBE decryption for each keyword in a matching record.

To implement the integrated index scheme, it is necessary for the server to maintain a table of

each keyword w that appears in any record in the log, the sequence number of the most recent record that contains w, and the corresponding value for j^*. Even with millions of different keywords, this does not impose a significant storage burden on the server. However, it leaks some information about one of the keywords stored in certain records, with a tendency to reveal more information about more recent records. Furthermore, for an investigator to take advantage of this index to start their search at the first relevant log entry, the server would need to know which keyword the investigator is looking for. Clearly, there is a need to have a better solution.

6.3 Anchor Records

The improved index structure is based on "anchor" records that appear at the end of each time zone. Each of these anchor records, which we denote as A_p, where p is the last time period of the time zone, contains part of the index relevant for the log records that occurred during that time zone. There are several tradeoffs involved in determining how often anchors should appear which are similar to the tradeoffs discussed in *Assumption 3*. Instead of storing a message in an anchor record, we store the sequence number of the previous anchor record. To let investigators access the previous anchor record efficiently, its sequence number is not encrypted.

Next, the keywords associated with A_p must be determined. First, we introduce some notation. Let W_{new} denote the set of recently used keywords, specifically, those used in at least one log record since the last anchor record. Let W_{old} be the set of keywords used only in records before the last anchor record. Finally, let s_w be the sequence number of the last log record to use keyword w. At a minimum, the keywords for R_i should be all of the keywords W_{new}. Specifically, for each $w \in W_{new}$, A_p will contain $c = IBE_w(flag|s_w|j^*)$. After A_p is written, s_w should be updated to equal A_p's sequence number, and each j^* updated accordingly. In other words, the next time a

record containing the keyword w is added to the log, its back pointer should be to this anchor record.

Next, we discuss the keywords from W_{old}. From a searching perspective, it is better to include each $w \in W_{old}$ just like those keywords in W_{new}, except that s_w should not be updated. This way, an investigator with access to any anchor record can immediately find the most recent log record containing the desired keyword. However, the storage requirements become prohibitively expensive since there can potentially be millions of distinct keywords in an entire log. Thus, we cannot include any of the keywords in W_{old} in the anchor record. To perform searches, investigators to need to traverse a chain of anchor records to find the first actual record that matches before following the back pointers. Some optimizations that allow the first matching record to be found quicker are discussed as future work.

We note that these anchor records are similar to the index described in [WBD04]. One difference between their construction and ours is that their index records contain all the information about the matching records within a time zone, but our anchor records only contain information about the previous matching record. However, the more significant difference is that their index records are independent for each time zone, while ours allow a connection from the first record in a time zone to the anchor record of the previous time zone containing a matching record. This connection allows for large stretches of time zones without any matching records to be skipped, which greatly improves our performance when looking for infrequently occurring keywords.

Chapter 7

SEALED

In the previous chapter, we discussed how to build an integrated index using anchor records to speed up searches. These anchor records allow an investigator to skip log records which are known not to contain the keyword she is searching for. Thus, the anchor records are used for efficiency, since records may be retrieved directly. However, if we change the index structure such that the only way to access log records is through the anchor records, then we can provide time restricted searches.

At a high level, we achieve efficient searches on encrypted audit logs as follows. As all log records can be retrieved by any entity, this necessitates that records be encrypted. In particular, regular log records are encrypted with a hash of themselves[6], and special anchor records, which occur once per time zone, and are encrypted with a time key and are signed with an evolving authentication key. These records serve as an index of the keywords that occurred in regular records during that time zone. Efficient searching is achieved by following "back pointers" to the previous matching record, which take the form of a hash of the record they point to. As a result, the decryption

[6] This technique is called convergent encryption, and has been used in other contexts to reduce the storage required when encrypting duplicate data with different keys; see [ABC02, DAB02].

key for a record is provided at the same time that a chain of authenticity is followed. Thus, only the signature of the final anchor record needs to be verified. Searches cannot extend further in time since anchor records for unauthorized time zones cannot be decrypted. To prevent an active adversary on the server from learning the decryption key for a record, log records are indexed by a hash of their key, and back-pointers are stored in an encrypted form.

7.1 Normal Log Records

Since the server can efficiently retrieve a log record R_i by more than just its sequence number (i), it can lookup an encrypted record R_i by its hash. However, this would allow the server to decrypt all records since it has access to the key (the hash) and the ciphertext. To fix this problem, the server looks up records by $h(h(R_i))$, which is the hash of the encryption key[7]. In this case, knowledge of the hash of a record is sufficient to retrieve that record from the server, decrypt it, and verify its integrity, without giving an adversary who has compromised the server means of doing the same. Furthermore, hashes of previous records can be included in later records (within backpointers) to form chains of records that can be authenticated via the final record in the chain. (This idea was previously illustrated in Figure 1.1.) We note that in this construction, the sequence number of a record is no longer required, but sequence numbers allow records from separate searches to be combined in sorted order. Specifically, the i-th regular record in a log can be constructed as:

$$R_i = \langle i, time(R_i), E_{K_i}(m_i), c_{i1}, \ldots, c_{i\ell_i} \rangle$$

where K_i is a randomly chosen key for record i, and each c_{ij} is an index fragment for a keyword that appears in record R_i. c_{ij} is derived from metadata maintained by the server. To illustrate how

[7] This idea is similar in nature to one derived independently by Kevin Fu [F04].

the metadata is updated, consider the last record X that contained keyword w. Let j^* denote the index fragment in X linked to w. Furthermore, let the indirection block of X for keyword w be $ib_w = \langle h(X), j^*, flag \rangle$. To enable efficient retrieval of records, a backpointer for w is created as $r_w = \left\langle K_w^{next}, E_{TK_{p'}}(ib_w) \right\rangle$ where $p' = time(X)$ and K_w^{next} is the key used to encrypt part of the subsequent record containing w (e.g, an index fragment in R_i). In this way, the decryption key for a record is provided at the same time that its chain of authenticity is followed. The stored metadata is K_w^{next} and $IBE_w(r_w)$. For a particular c_{ij} the server can derive this index fragment as $c = \left\langle E_{K_w^{next}}[K_i], IBE_w(r_w) \right\rangle$. For more details about how a log record is added, see Algorithm 1.

7.2 Anchor Records

If an investigator obtains r_w from the server's metadata, she can proceed with a search starting from the present. In order to meet *Requirement 2*, it must be possible to begin a search from any time zone in the past. To allow for such searches, special "anchor" records (occurring once at the end of each time zone) can serve as an index of the keywords that appeared during each time zone. Such anchor records can be constructed as:

$$A_p = \left\langle p, h(A_{p-1}), c_{p1}, \ldots, c_{p\ell_p} \right\rangle$$

Each c_{pj} is simply the $IBE_w(r_w)$ that is stored as metadata. After A_p is written, this metadata is updated in the same way as for a normal record. More details about adding anchor records are in Algorithm 2. Note that this metadata is encrypted, so an adversary who compromises the server before A_p is written will not be able to decrypt the log records for the current time zone. This is a direct contrast to the indexing optimization of [WBD04] in which the keys for records in the current block are kept on the server until the index record is written.

The hash of the previous anchor record is included to make authentication easier. If A_p is authenticated, $h(A_{p-1})$ contained within A_p is also authentic, so it may be used to authenticate A_{p-1}. Consequently, all previous anchor records can be authenticated is a similar manner. Anchor records can be digitally signed using an evolving key to achieve forward security. For delegated searches, the investigator only needs to verify the signature of the most recent anchor record to authenticate the results of an entire search, since all previous anchor records can be authenticated with the hash chain. Similarly, once the backpointer to the first matching record is found within an authenticated anchor record, the back pointers form another hash chain to authenticate all matching records.

Searches can be prevented from extending past anchor records for which the investigator is not given access to by encrypting anchor records with time-zone specific keys; thus anchor records for unauthorized time zones cannot be decrypted. As discussed, normal log records are also encrypted. The server simply encrypts R_i and A_p and indexes them by a hash of their respective encryption keys. Specifically, the server now stores regular and anchor records as:

$$\begin{aligned} R'_i &= \left\langle h(h(R_i)), E_{h(R_i)}(R_i) \right\rangle \\ A'_p &= \left\langle h(TK_p), E_{TK_p}(A_p), SIG_{AK_p^-}(A_p) \right\rangle \end{aligned}$$

respectively. $SIG_{AK_p^-}(m)$ denotes a DSA signature [F186] using AK_p^- as the private key[8]. Recall AK_p is derived in a tree-based manner similar to time-based keys (see Section 5.3) and hence AK_p^- is known by the server. While it may be argued that more key material is now stored on the server, our tree based construction limits the exposure of future keys when the server is compro-

[8] AK_p^- is derived as $\{h(AK_p)|h(AK_p+1)\}$ mod q, where q is a 160-bit prime.

mised. This is a direct contrast to the schemes of [KS99, WBD04], where all future versions of the linearly derived authentication key are exposed when the server is compromised (see *Requirement 10*).

7.3 Protocols

7.3.1 SETUP

Alice randomly chooses a master time key (TK_*) and a master authentication key, (AK_*). Next, she initializes an instance of the IBE scheme (see [BF01]), and generates DSA parameters (see [F186]). The parameters are published and provided to the server. Alice retains the master IBE secret, the master time key, and the master authentication key. As discussed earlier (see *Assumption 3*), periodic interaction is required between Alice and the server to update the key material on the server. This interaction is similar to DELEGATION, except that authentication keys are also provided to the server.

7.3.2 DELEGATION

Suppose an investigator requests to search for records that occurred during the set of time intervals T that are related to any keywords in W. If Alice approves the investigator's request, she can provide several keys to the investigator to facilitate the search as follows.

1. First, for each keyword $w \in W$, Alice uses the master IBE secret to generate the IBE decryption key, d_w.

2. Next, for each disjoint interval of time $[p_{start}, p_{stop}]$ in T, Alice uses AK_* to derive $AK_{p_{stop}}$, and provides the public key $AK^+_{p_{stop}} = g^{\{h(AK_{p_{stop}})|h(AK_{p_{stop}}+1)\}}$ modulo a 1024 bit prime

(i.e., the DSA modulus).

3. Finally, for each time interval $[p_{start}, p_{stop}]$ in T, Alice uses TK_* to provide a set of time keys that allows an investigator to compute TK_p for each $p \in [p_{start}, p_{stop}]$.

Both time and authentication keys are derived from a tree structure as described in Section 5.3. The set of keys that must be transmitted to the investigator corresponds to the minimum set of nodes in the tree whose subtrees span the entire set of leaves between p_{start} and p_{stop}, but does not span any other intervals. An example was presented in Section 5.3; for more details, see Algorithm 3.

7.3.3 SEARCHING AND DECRYPTION

For ease of exposition, we discuss searching and decryption in terms of a single keyword w and time interval $[p_{start}, p_{stop}]$ (It should be obvious to the reader that searches for multiple keywords or time periods may be performed in parallel.) An overview of the search and decryption procedure is provided in Figure 7.1.

If an investigator wishes to retrieve a record using a key K, she provides $h(K)$ to the server, which in turn retrieves the corresponding content (e.g., R'_i or A'_p). At this point, the investigator can now decrypt that content with K to obtain R_i or $\left\langle A_p, SIG_{AK_p^-}(A_p) \right\rangle$, respectively.

SEARCHING FOR THE FIRST MATCH

To locate the first matching record within the scope of her search, an investigator starts by deriving $TK_{p_{stop}}$, and using it to retrieve $A_{p_{stop}}$ and its signature. She verifies that signature using $AK^+_{p_{stop}}$; if the signature does not match or the the server fails to return the record, the search is terminated. In this case, the investigator deems that the server is misbehaving or the log has been

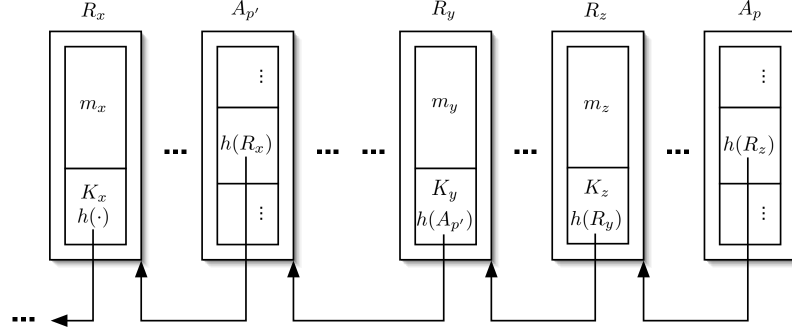

Figure 7.1: A view of a search spanning multiple time zones. In regular records the message m_i is encrypted with K_i, the index fragment is IBE encrypted for the keyword being searched for, and the entire record is encrypted with a hash of itself. Anchor records are encrypted with time keys. A search is performed by following the pointers within index fragments – these fragments contain the key to decrypt the current record, and a hash of the previous matching record.

altered. Otherwise, if the signature is verified, the investigator has now authenticated $A_{p_{stop}}$. Now, let $A = A_{p_{stop}}$ and $p = p_{stop}$.

Recall that each index fragment c in A is derived from a distinct keyword. For ease of notation, suppose that c was derived from w'. That is, $c = \left\langle E_{K^{next}_{w'}}[K], IBE_{w'}(r_{w'})\right\rangle$. We emphasize that w' is unknown to the investigator. Hence, given A and p, the investigator can search for the first match as follows:

1. For each c in A she computes $r_w = IBD_{d_w}(IBE_{w'}(r_{w'}))$.

2. If $w \neq w'$, then this decryption will not be successful. If none of the index fragments can be decrypted, then the prior anchor record must be retrieved. Otherwise, r_w can be used as input to the SEARCHING FOR SUBSEQUENT MATCHES algorithm.

3. Retrieve the next anchor record by deriving TK_{p-1} and using it to retrieve A_{p-1}. Next, compare the hash of A_{p-1} to that stored in A. If these hashes are not in agreement, then the search is terminated, and tampering has occurred. Otherwise, the authenticity of A_{p-1} is now established since its hash was previously authenticated.

4. Let $A = A_{p-1}$ and $p = p - 1$. Repeat Steps 1-3 until a decryption is successful or $p = p_{start}$.

If $p = p_{start}$ and no decryptions were successful, then no matching records exist, and the search is complete. Otherwise, r_w can be used as input to the next algorithm, which retrieves all matching records. Note that since A authentic, and r_w is contained within A, then it is also authentic. For a more detailed description of how to find the first matching record, see Algorithm 4.

SEARCHING FOR SUBSEQUENT MATCHES

Assume that the investigator has decrypted and authenticated r_w from an index fragment $c = \left\langle E_{K_w^{next}}[K], IBE_w(r_w) \right\rangle$ from a record during time period p, where $r_w = \left\langle K_w^{next}, E_{TK_{p^*}}(ib_w) \right\rangle$. Here, p^* is the time period during which the previous matching record occurred. If the current record is not an anchor record, then K_w^{next} can be used to decrypt $E_{K_w^{next}}[K]$. Next, K can be used to decrypt m. To obtain the next matching record, the investigator proceeds as follows:

1. Decrypt the encrypted indirection block $E_{TK_{p^*}}(ib_w)$ by sequentially trying each time key $TK_{p'}$ (for $p' \in [p_{start}, p]$) in descending order. If a decryption with $TK_{p'}$ is correct (e.g., $p' = p^*$), then $flag$ is recovered successfully. However, if no decryptions are successful, no more matching log records exist within the interval $[p_{start}, p_{stop}]$, and the search is complete.

2. Let the result of the decryption be $ib_w = \langle h(X), j^*, flag \rangle$, where X is the previous record related to w, and j^* is which index fragment in X is related to w.

3a. If $p^* < p$, then X is an anchor record. In this case, the investigator derives TK_{p^*}, and uses it to retrieve A_{p^*}. The authenticity of A_{p^*} can be established by comparing a hash of the returned record to that of $h(X)$ in ib_w. If these hashes do not agree, then the search is terminated, and tampering has been detected. Otherwise, r_w can be obtained by following

Step 1 in the algorithm to find the first match, where only the j^*-th index fragment in A_{p^*} is considered. r_w can then used as input in Step 1.

3b. If $p^* = p$, then X is a normal log record, and $h(X)$ is the encryption key used to retrieve X. Since r_w was authenticated, and ib_w is a part of r_w, ib_w is also authenticated. Thus, since ib_w contains $h(X)$, if the hash of the retrieved record agrees, X is authenticated. The next r_w can be obtained by IBE decrypting the appropriate part of the j^*-th index fragment in X. Repeat Step 1 with the new r_w (and $p = p^*$).

For a more detailed description of how to retrieve subsequent matches, see Algorithm 5.

Chapter 8

Performance

In this chapter we evaluate the performance of our proposed technique for searching encrypted audit logs. All experiments were performed on a dual 1.3 GHz G4 server with 1 GB of memory. For symmetric key operations we used the Openssl [SSL] implementation of AES with 128-bit keys [F197]. 160 bit SHA-1 was used as our hash function [F180], and DSA with a 1024 bit modulus and 160 bit secrets for the digital signature operations [F186]. We used the Stanford Identity-Based encryption library [IBE] for IBE operations with a 512-bit prime and a subgroup of size 160-bits.

To empirically evaluate the performance of our scheme, we conducted several experiments. First, the realtime costs associated with inserting encrypted log entries is evaluated by replaying a log depicting activity recorded by the Snort IDS [CBF04] system running on a Mac OS X server during a period spanning several months. The events in the log occur in bursts, with half the alerts occurring in a 10 day span. Of the days with at least one alert, there were an average of 260 alerts per day, but a median of only 17. During the observed period, roughly 27,000 alerts were recorded, including attacks from nearly 1,700 distinct IP addresses, with roughly 500 of these IP involved in more than 10 alerts. On average, records were described by 6 keywords. The encrypted log itself

required 62.5 MB of storage with the average record size being 2.3 Kbytes.

Unfortunately, due to its bursty nature, the log is not as diverse as one would expect, and not well suited for testing the theoretical performance of our scheme. To address this, we evaluated the theoretical performance characteristics[9] on a synthetic log with characteristics similar to that of the Snort log. The synthetic log contains 30,000 entries timed with an exponential distribution. The expected elapsed time between records was 15 minutes, which results in a log spanning roughly 11 months. To make the synthetic log more closely approximate the characteristics of the Snort log, records are described with an average of 5 keywords that include one of 10 random attack types, one of 500 randomly chosen IP addresses, keyword ALL, and keywords that appear with known probabilities. Table 8.1 depicts the cost to insert, in real time, all the records from an synthetic log to a MySQL database. These results are averaged over five runs. The table shows, for example, that if log contain records with an average of 6 keywords, throughput is roughly 15 records per second. We note that this throughput corresponds to the maximum peak rate of 15 records per second observed in the Snort traces[10].

Next, to evaluate the efficiency of our approach we experiment with searches on both the real and synthetic logs. For the synthetic log, searches are performed for keywords occurring with varying probability. The left-hand side of Figure 8.1 depicts the results of searches averaged over 100 randomly generated logs. The results show that when searching for keywords that appears

[9] Under the assumption of a uniform distribution of keywords in the log, the number of IBE decryptions needed to perform a search for a keyword w occurring with probability x is as follows. If r records occur per time zone, the probability of a record with w occurring within a particular time zone is $y = 1-(1-x)^r$. The expected number of anchor records to be searched before the first match is found is $1/y$. Thus, the total number of IBE decryptions is one for each distinct keyword in each of the $1/y$ anchor records until the first match is found, one per subsequent anchor record containing a matching record, and one per matching record.

[10] According to [WBD04], the MIRACL package [MIR] has an implementation of IBE that is twice as fast as [IBE], and if this is the case, we can expect higher throughput.

words/record	time (s)	records/sec	words/record	time (s)	records/sec
1.0	343.5	87.3	3.0	879.0	34.2
1.5	482.7	62.2	4.0	1168.2	25.7
2.0	617.4	48.6	5.0	1509.9	19.9
2.5	749.0	40.0	6.0	2069.7	14.5

Table 8.1: Running times in seconds to import 30,000 records with varying numbers of keywords per record.

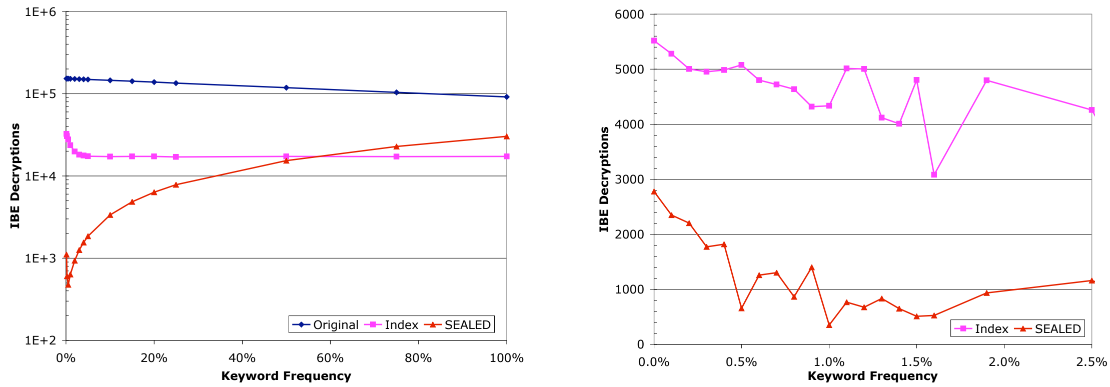

Figure 8.1: Incurred IBE decryptions as a function of the keyword frequency. (Left) Searching on the synthetic logs averaged across 100 runs. (Right) Searching on the Snort log.

infrequently, our technique achieves significantly better performance penalty than that of [WBD04]. For example, if the search keyword occurred within 5% of the records in the log, our technique results in roughly 10 times fewer IBE decryptions than the enhanced index scheme of [WBD04]. Moreover, if the keyword being searched appears in less than 1% of the records (which might be the case if searching for an IP address for example), our approach requires roughly forty times less operations than the enhanced scheme of [WBD04]. In one particular case, when searching for a keyword that only appeared in one record at the end of the log, our scheme only required 12 decryptions, while [WBD04] required roughly 6,000 decryptions.

To see why this is the case, let n denote the number of entries in the log, b the number of log entries per block, u the average number of distinct keywords in a block, and k the number of log

records that are related to a keyword w. To perform a search for w using the scheme of [WBD04] an investigator must retrieve $O(n/b + k)$ log records and perform $O(u \cdot \frac{n}{b})$ IBE decryptions. In practice, since most useful searches are for infrequently occurring keywords (e.g, an IP address) then $n/b \gg k$, which means that there are many distinct keywords in each block and so $u > b$. Hence, the number of records to retrieve (and associated IBE operations to be performed) becomes proportional to the total number of records (n), rather than the optimal value (i.e., the number of *matching* records, k). By contrast, our scheme is similar in performance to that of [WBD04] up until the first matching record is found, after which the number of retrieved records and IBE decryptions are both $O(k)$. Thus, the our biggest performance advantage is achieved when the first matching record occurs toward the end of the search interval.

Similar performance improvements are observed for searches on the Snort log itself (shown in the right-hand side of Figure 8.1). Here, each point in the graph depicts the average number of IBE decryptions required to separately search for each keyword that appears in the specified percent of records. While the number of decryptions required when searching the Snort alerts for keywords that appeared with $< 1\%$ frequency results in only twice as few IBE decryptions as the enhanced index scheme, the vast majority of records in the log are clustered to one region—in particular, towards the beginning of the log and therefore is not reflective of the average case scenario. In practice, if searches are limited to a more well defined period around the activity in the log, then we can expect performance improvements similar to that observed for the synthetic logs. Nonetheless, we believe our constructions significantly advance the practicality of searches on encrypted audit logs to date.

Chapter 9

Conclusion and Future Work

In this thesis we presented the construction of an encrypted audit log that allows its owner to efficiently delegate searches that are limited by both time and content. We explored several problems. First, we investigated how an audit log can be searched during an investigation after the server is compromised, and how the exposure of private information stored in the log can be limited during a search. Furthermore, we examined how the owner of the log can delegate searches on the log to authorized third parties efficiently. These delegations may be voluntary (for example, to a managed security company to investigate an attack) or involuntary (for example, to comply with a search warrant). Delegated searches are limited in scope and duration, provide a guarantee of completeness, and are efficient.

Moreover, we showed that investigators can efficiently find all relevant records, and can authenticate retrieved records without interacting with the owner of the log. In addition, we provided an empirical evaluation of our techniques using IDS alerts collected over a span of a few months. We showed that our constructions are practical and improve the state of the art. As shown in Chapter 8, our improvements over prior work manifest themselves when searching for infrequently occurring

keywords.

An interesting direction for future work is in exploring ways to decrease the cost associated with finding the first matching record for the keyword being searched. One such approach is to make use of Bloom filters [B70] based on the keywords that appeared in the anchor record of each time zone. In doing so, needless IBE decryptions can be avoided. However, this approach has a potential security weaknesses, in that an investigator could use the Bloom filter to determine if a keyword appeared in an anchor record, even without permission to do so. As such, other approaches for finding the first matching record more efficiently need to be explored.

An alternate direction of future work could involve relaxing the assumption that the owner of a log is not subject to compromise. If that is the case, then work on separation of trust, for example the work on Intrusion Resilient Signatures [IR02] and Capture Resilient Devices [MR01], may be a fruitful avenue to pursue.

Appendix A

Algorithm Details

A.1 LOG

There are two parts to the protocol of adding a log record. First, in Algorithm 1, the message itself must be added to the log. Once the time zone changes, Algorithm 2, which creates the anchor record and updates the metadata, must be executed. Note that for each keyword w that has appeared in the log, $IBE_w(r_w)$ and K_w^{next} are stored in the database. Also, note that by $prev(p)$, we mean the last time period in the previous time zone, which is normally just $p - 1$.

A.2 DELEGATE

This protocol allows Alice to determine the set of time keys needed for any interval of time zones. The details are in Algorithm 3, but some terminology is needed. We define two functions (ext_x and $common$) on full 32-bit time periods. For a single 32-bit time period a, we denote $ext_x(a) = p$ as the shortest prefix of a such that the remaining $32 - |p|$ bits of a are all x's (for some bit x). In other words, it returns a prefix p of a that excludes all trailing x's. From the

1: INPUT: m, a message to put in the log (during time period p), and TK_p, the current time key
2: OUTPUT: R', the encrypted log entry
3: $K \in_R \{0,1\}^*$
4: $R \leftarrow \langle i, p, E_K(m) \rangle$
5: **for all** $w \in words(m)$ **do**
6: **if** $current(w) = undefined$ **then**
7: $ib_w \in_R \{0,1\}^*$
8: $K_w^{next} \in_R \{0,1\}^*$
9: $r_w \leftarrow (K_w^{next}, ib_w)$
10: Store $K_w^{next}, IBE_w(r_w)$
11: **end if**
12: $c \leftarrow \left\langle E_{K_w^{next}}(K), IBE_w(r_w) \right\rangle$
13: $R \leftarrow \langle R, c \rangle$
14: **end for**
15: $j \leftarrow 0$
16: **for all** $w \in words(m)$ **do**
17: $ib_w \leftarrow \langle h(R), j, flag \rangle$
18: $K_w^{next} \in_R \{0,1\}^*$
19: $r_w \leftarrow \langle K_w^{next}, E_{TK_p}(ib_w) \rangle$
20: $current(w) \leftarrow true$
21: Store $K_w^{next}, IBE_w(r_w)$
22: $j \leftarrow j + 1$
23: **end for**
24: **return** $R' = \left\langle h(h(R)), E_{h(R)}[R] \right\rangle$

Algorithm 1: Insertion of messages to the audit log.

1: INPUT: p, the current time period, AK_p, the time period's authentication key, and TK_p, the time period key
2: USES: $h(A_{prev(p)})$, the hash of the previous anchor record.
3: OUTPUT: A' the encrypted anchor record for time period p
4: $A \leftarrow \left\langle p, prev(p), h(A_{prev(p)}) \right\rangle$
5: **for all** $\{w : current(w) = true\}$ **do**
6: Load $IBE_w(r_w)$
7: $A \leftarrow \langle A, IBE_w(r_w) \rangle$
8: **end for**
9: Store $h(A)$
10: $j \leftarrow 0$
11: **for all** $\{w : current(w) = true\}$ **do**
12: $ib_w = \langle h(A), j, flag \rangle$
13: $K_w^{next} \in_R \{0,1\}^*$
14: $r_w \leftarrow \langle K_w^{next}, E_{TK_p}(ib_w) \rangle$
15: $current(w) \leftarrow false$
16: Store $K_w^{next}, IBE_w(r_w)$
17: $j \leftarrow j + 1$
18: **end for**
19: $A' = \left\langle h(TK_p), E_{TK_p}(A), SIG_{AK_p^-}(A) \right\rangle$
20: **delete** AK_p and TK_p
21: $p \leftarrow p + 1$
22: **return** A'

Algorithm 2: Procedure for advancing to the next time zone.

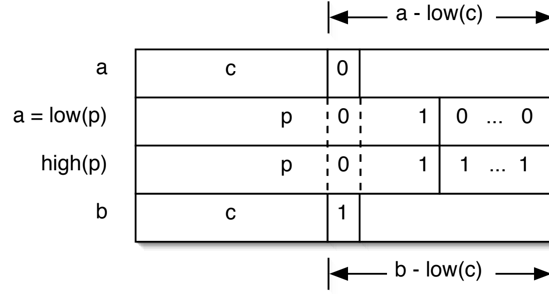

Figure A.1: Providing time key TK_p (where $p = exp_0(a)$) will allow all time keys between $a = low(p)$ and $high(p)$ to be derived. Then, the algorithm can be run again, with $a' = high(p) + 1 \in (a, b]$.

definitions, it should be clear that $low(ext_0(a)) = a$ and $high(ext_1(a)) = a$. For two 32-bit time periods, a and b, we denote $common(a, b) = c$ as the longest common prefix that a and b share, $|c| \in [0, 32]$.

In designing our algorithm, we desire that Alice be permitted derive the time keys to generate a range of time periods, from a to b, inclusive. A time key TK_p is safe if it does not allow the recipient to derive any time keys out of range, specifically, $safe(TK_p) = low(p) \geq a \cap high(p) \leq b$. A set of time keys $S = \{TK_{p_1}, \ldots, TK_{p_k}\}$ is covering if it allows the generation of all time keys $\in [a, b]$. If all of the time keys in S are safe, then the generation is exactly the set of time keys $\in [a, b]$. For efficiency, the smallest number of time keys ($|S|$) should be returned.

In a particular instance of the algorithm, a is the starting period and b is the stopping period ($a \leq b$, both inclusive). The common prefix of a and b is c, as shown in Figure A.1. The remaining bits of a and b can be computed by subtracting $low(c)$ from them. If a and b are aligned to cover an entire binary tree of height $32 - |c|$, then TK_c can be returned, from which all $2^{32-|c|}$ time keys in $[a, b]$ can be generated. Most of the time, this alignment will not be possible, thus we provide time keys that allow us to increase a to a' and decrease b to b' such that $[a', b']$ is closer to being well aligned.

We obtain a' as follows. If the remainder of a is all 0's (e.g. $a - low(c) = 0$), then we only need to adjust b, and $a' = a$. Otherwise, we observe how many trailing 0's are in a, and take the prefix $p = ext_0(a)$, as indicated in Figure A.1. Since $a - low(c) > 0$, there must be at least one non-zero bit in the last $32 - |c|$ bits of a, thus, $|p| > |c|$. Furthermore, the last bit of p must be a 1. Providing TK_p will reveal all time keys from period $low(p) = a$ through $high(p)$. Since $|p| > |c|$, it follows that $high(p) < b$. Thus, TK_p is safe, and we can continue the algorithm on a reduced range of time periods. We let $a' = high(p) + 1$, derive b' in a similar manner, and run the algorithm on $[a', b]$. Note that the form of a' is to have a prefix c, and to end with at least one more zeros than a, since the last bit of p is a 1. Thus, in the next iteration, $|p|$ will decrease (until it reaches $|c|$; when that happens, the algorithm can return a final key and return).

```
1:  DISTRIBUTE-KEYS(a, b)
2:  INPUT: The ranges of a search starting at a and stopping at b (a ≤ b).
3:  OUTPUT: A minimal safe covering set S of [a, b]
4:  S ← ⟨⟩
5:  c ← common(a, b)
6:  return S = {TK_c} if low(c) = a ∩ high(c) = b
7:  if low(c) = a then
8:      a' ← a
9:  else
10:     p ← ext_0(a)
11:     S ← S ∪ {TK_p}
12:     a' ← high(p) + 1
13: end if
14: if high(c) = b then
15:     b' ← b
16: else
17:     p ← ext_1(b)
18:     S ← S ∪ {TK_p}
19:     b' ← low(p) − 1
20: end if
21: return S if a ≠ a' ∩ b ≠ b' ∩ a' − 1 = b'
22: return S = S ∪ DISTRIBUTE-KEYS(a', b')
```

Algorithm 3: Distribution of time keys.

A.3 SEARCH

An investigator must obtain secrets from Alice using the delegation protocol before she can perform a search. Searching is broken up into two phases. In the first phase, anchor records are scanned until the first one containing a matching record is found. Then, in the second phase, all subsequent matching records are returned. For more details, see Algorithms 4 and 5, respectively. These algorithms are presented for the simple case of searching for one keyword in one continuous interval of time, but can easily be extended to search for multiple keywords in multiple time zones.

Both of these phases make use of another procedure: given a request for a log record, by the hash of its encryption key, the server returns the encrypted part of the log record (and for anchor records, also the signature), or an error message that no such entry exists. We use the function $\mathrm{getrec}(x)$ to mean the execution of an instance of this protocol.

1: To perform a search for keyword w over an interval of time $T = [p_1, \ldots, p_k]$:
2: INPUT: d_w, the IBE secret for w, $AK^+_{p_k}$, and the time keys for each $p \in T$ (provided by Alice from the delegation protocol)
3: OUTPUT: The set of messages in the log records $match(T, W)$
4: $p \leftarrow p_k$
5: $next_hash \leftarrow 0$
6: **while** $p \geq p_1$ **do**
7: $key \leftarrow TK_p$
8: $A' \leftarrow \langle x = E_{TK_p}(A), sig \rangle \leftarrow \text{getrec}(h(key))$
9: **abort if** error retrieving A'
10: $A \leftarrow D_{TK_p}(x)$
11: $\langle p, p_{prev}, next_hash', c_1, \ldots, c_j \rangle \leftarrow A$
12: **abort if** $p = p_k \cap$ cannot verify sig with $AK^+_{p_k}$
13: **abort if** $p < p_k \cap next_hash \neq h(A)$
14: $next_hash \leftarrow next_hash'$
15: **for all** $c = IBE_{w'}(r_{w'}) \in \{c_i, \ldots c_j\}$ **do**
16: $z = E_{TK_p}(ib_w) \leftarrow IBD_{d_w}(c)$
17: **next if** decryption failed (e.g., $w \neq w'$)
18: $\langle next_h, j^*, flag \rangle \leftarrow D_{TK_p}(z)$
19: **abort if** decryption failed
20: Run Phase 2 with extra inputs $p, next_h, j^*$ and return result
21: **end for**
22: $p \leftarrow p_{prev}$
23: **end while**
24: **return** no records match

Algorithm 4: Finding the most recent time zone with a keyword matching the search criteria.

1: To perform a search for keyword w over an interval of time $T = [p_1, \ldots, p_k]$:
2: INPUT: d_w, the IBE secret for w and the time keys for each $p \in T$ (provided by Alice), p the most recent time period to contain a record of interest, $next_h$, a hash of the first record of interest, and j^*, which index fragment in that record is relevant.
3: OUTPUT: The set of messages in the log records $match(T, W)$
4: $S \leftarrow \langle\rangle$
5: **while** $p \geq p_1$ **do**
6: $\quad R' \leftarrow \text{getrec}(h(next_h))$
7: $\quad$ **abort if** error retrieving R'
8: $\quad R \leftarrow D_{next_h}(R')$
9: $\quad$ **abort if** $h(R) \neq next_h$
10: $\quad \langle i, p, msg = E_K(m), c_1, \ldots, c_j\rangle \leftarrow R$
11: $\quad \left\langle x = E_{K_w^{next}}(K), y = IBE_w(r_w)\right\rangle \leftarrow c_{j^*}$
12: $\quad \left\langle K_w^{next}, z = E_{TK_{p^*}}(ib_w)\right\rangle \leftarrow IBD_{d_w}(y)$
13: $\quad K \leftarrow D_{K_w^{next}}(x)$
14: $\quad S \leftarrow \langle S, D_K(msg)\rangle$
15: $\quad p' = p$
16: $\quad$ **while** $p' \geq p_1$ **do**
17: $\quad\quad \langle next_h, j^*, flag\rangle \leftarrow D_{TK_{p'}}(z)$
18: $\quad\quad$ **break if** decryption successful
19: $\quad\quad p' \leftarrow p' - 1$
20: $\quad$ **end while**
21: $\quad$ **return** S **if** no decryptions were successful
22: $\quad$ **if** $p' < p$ **then**
23: $\quad\quad p \leftarrow p'$
24: $\quad\quad key \leftarrow TK_p$
25: $\quad\quad A' = \langle b = E_{TK_p}(A), sig\rangle \leftarrow \text{getrec}(h(key))$
26: $\quad\quad$ **abort if** error retrieving A'
27: $\quad\quad A \leftarrow D_{key}(b)$
28: $\quad\quad$ **abort if** $h(A) \neq next_h$
29: $\quad\quad \langle \ldots, c_1, \ldots, c_j\rangle \leftarrow A$
30: $\quad\quad tmp = E_{TK_p}(ib_w) \leftarrow IBD_{d_w}(c_{j^*})$
31: $\quad\quad \langle next_h, j^*, flag\rangle = D_{TK_p}(tmp)$
32: $\quad$ **end if**
33: **end while**
34: **return** S

Algorithm 5: Finding all subsequent matching records.

Bibliography

[ABC02] A. Adya, W. Bolosky, M. Castro, R. Chaiken, G. Cermak, J. Douceur, J. Howell, J. Lorch, M. Theimer and R. Wattenhofer. FARSITE: Federated, Available, and Reliable Storage for an Incompletely Trusted Environment. In Proceedings of 5th Symposium on Operating Systems Design and Implementation, Dec 2002.

[BBD01] M. Bellare, A. Boldyreva, A. Desai and D. Pointcheval. Key-privacy in Public-Key Encryption. Extended abstract in Advances in Cryptology – Asiacrypt 2001 Proceedings, Lecture Notes in Computer Science Vol. 2248, C. Boyd ed, Springer-Verlag, 2001.

[BC04] S. M. Bellovin and W. R. Cheswick. Privacy-Enhanced Searches Using Encrypted Bloom Filters. Cryptology ePrint Archive, Report 2004/022, 2004.

[B70] B. Bloom. Space/time Trade-offs in Hash Coding with Allowable Errors. Communications of the ACM, 13(7):422–426, Jul 1970.

[BCO03] D. Boneh, G. Di Crescenzo, R. Ostrovsky, and G. Persiano. Public Key Encryption with Keyword Search. Cryptology ePrint Archive, Report 2003/195, 2004.

[BF01] D. Boneh and M. Franklin. Identity Based Encryption from the Weil Paring. In Proceedings of CRYPTO 2001, Lecture Notes in Computer Science, Vol. 2139, Springer-Verlag,

pp. 213-229, 2001.

[BGL03] D. Boneh, C. Gentry, B. Lynn, and H. Shacham. Aggregate and Verifiably Encrypted Signatures from Bilinear Maps. In Advances in Cryptology – EUROCRYPT 2003 (E. Biham, ed.), Lecture Notes in Computer Science, International Association for Cryptographic Research, Springer-Verlag, Berlin Germany, 2003.

[CBF04] B. Caswell and J. Beale and J. Foster and J. Faircloth. Snort 2.0 Intrusion Detection System. Syngress, May, 2004. See `http://www.snort.org`

[CM04] Y. Chang and M. Mitzenmacher. Privacy Preserving Keyword Searches on Remote Encrypted Data. Cryptology ePrint Archive, Report 2004/051, 2004.

[CGN97] B. Chor and N. Gilboa and M. Naor. Private Information Retrieval by Keywords. TR CS0917, Department of Computer Science, Technion, 1997.

[CGK95] B. Chor, O. Goldreich, E. Kushilevitz and M. Sudan. Private Information Retrieval. In Proceedings of IEEE Symposium on Foundations of Computer Science, 1995, pages 41-50.

[DAB02] J. R. Douceur, A. Adya, W. J. Bolosky, D. Simon, M. Theimer. Reclaiming Space from Duplicate Files in a Serverless Distributed File System. In Proceedings of 22nd International Conference on Distributed Computing Systems, Jul 2002.

[F197] Federal Information Processing Standards. Advanced Encryption Standard (AES) – FIPS 197, November, 2001.

[F186] Federal Information Processing Standards. Digital Signature Standards (DSS) – FIPS 186, May, 1994.

[F198] Federal Information Processing Standards. The Keyed-Hash Message Authentication Code (HMAC) – FIPS 198, March, 2002.

[F180] Federal Information Processing Standards. Secure Hash Standard – FIPS 180-1, May, 1993.

[F04] K. Fu. Personal communication, Feb 2004.

[G04] E. Goh. Secure Indexes. Cryptology EPrint Archive, Report 2003/216, 2003.

[HS90] S. Harber and W. Stornetta. How to Time-Stamp a Digital Document. In A. Menezes and S. A. Vanstone, editors, Proceedings of CRYPTO 90, pages 437-455. Springer-Verlag, 1991. Lecture Notes in Computer Science No. 537.

[IR02] G. Itkis and L. Reyzin. SiBIR: Signer-Base Intrusion-Resilient Signatures. Cryptology ePrint Archive, Report 2002/054, 2002.

[JMS02] R. Johnson, D. Molnar, D. Song, and D. Wagner. Homomorphic Signature Schemes. In Proceedings of the RSA Security Conference, Cryptographers Track, February 2002.

[KS99] J. Kelsey and B. Schneier. Minimizing Bandwidth for Remote Access to Cryptographically Protected Audit Logs. In Proceedings of the 2nd International Workshop on Recent Advances in Intrusion Detection (RAID'99).

[L03] D. Lazarus. A Tough Lesson on Medical Privacy: Pakistani Transcriber Threatens UCSF Over Back Pay. San Francisco Chronicle, October 22, 2003.

[MR01] P. MacKenzie and M. K. Reiter. Networked cryptographic devices resilient to capture. In Proceedings of the 2001 IEEE Symposium on Security and Privacy, pages 12–25, May 2001.

[M01] W. Mao. Timed-release cryptography. Selected Areas in Cryptography VIII (SAC'01), Toronto, Ontario, Canada, August 2001, Lecture Notes in Computer Science, Vol 2259, Springer, 2001, pages 342-357.

[MHS03] M. Mont, K. Harrison, and M. Sadler. The HP Time Vault Service: Exploiting IBE for Timed Release of Confidential Information. In Proceedings of WWW 2003, Security and Privacy Track, May 2004, Budapest, Hungary.

[MNT04] E. Mykletun, M. Narashimha and G. Tsudik. Authentication and Integrity in Outsourced Databases. To appear in The 11th Annual Network and Distributed System Security Symposium.

[SSL] Openssl Project. OpenSSL. See `http://www.openssl.org`.

[SW04] A. Sahai and B. R. Waters. Fuzzy Identity Based Encryption. Cryptology ePrint Archive, Report 2004/086, 2004.

[SK98] B. Schneier and J. Kelsey. Cryptographic Support for Secure Logs on Untrusted Machines. In Proceedings of the 7th USENIX Security Symposium (San Antonio, TX, USA, Jan. 1998), pp. 53-62.

[MIR] Shamus Software Ltd. MIRACL: Multiprecision Integer and Rational Arithmetic C/C++ Library. `http://indigo.ie/~mscott`.

[SWP00] D. Song, D. Wagner, and A. Perrig. Practical Techniques for Searches on Encrypted Data. In Proceedings of 2000 IEEE Symposium on Security and Privacy, May 2000.

[IBE] Stanford Applied Cryptography Group. IBE Secure Email. See `http://crypto.stanford.edu/ibe`.

[WBD04] B. R. Waters, D. Balfanz, G. Durfe, and D. K. Smetters. Building an Encrypted and Searchable Audit Log. To appear in The 11th Annual Network and Distributed System Security Symposium.

About The Author

Darren Davis was born in Brooklyn, New York, on December 23, 1981. He graduated from Marlboro High School in Marlboro, New Jersey, in June 1999. After high school, he joined Johns Hopkins University in Baltimore, Maryland, as a Computer Science major. During the Fall semester of 2002, he was accepted into the Concurrent Bachelors/Masters Program in the Computer Science Department. In May 2003, he graduated with a Bachelor of Science in Computer Science and a secondary major in Mathematical Sciences and a minor in Entrepreneurship and Management. He was awarded a Master of Science in Engineering Degree in Computer Science in May 2004.