

Secure Logging of Data Transferred from Optical Disc Media

Bachelor Thesis

Björn Michael Pantel | 1388718

Informatik



TECHNISCHE
UNIVERSITÄT
DARMSTADT



Fraunhofer

SIT

Björn Michael Pantel
Matrikelnummer: 1388718
Studiengang: Bachelor Informatik

Bachelor Thesis
Thema: "Secure Logging for Data Transfer from Optical Disc Media"

Eingereicht: 4. August 2011

Betreuer: Dr. Martin Steinebach
York Yannikos

Prüfer: Prof. Dr. Michael Waidner
Fraunhofer-Institut für Sichere Informationstechnologie

Abstract

German authorities who confiscate optical media use an automatic copying machine to duplicate the data onto external hard drives. This allows for efficient analysis of the data. In the current mode of operation, the copied data is used as evidence during the course of internal investigations, lawsuits, government investigations, audits, and other formal matters. However, any party having access to the drive can modify data which resides on it. It is not possible to verify the integrity and authenticity of the copied data.

This bachelor thesis describes a concept that enhances the functionality of the copying machine by utilizing secure logging with the intent of providing digital evidence. This concept incorporates state of the art secure logging approaches, and is resistant to the copying and verification attacks described in this document. In addition, a Java based prototype is implemented, as part of this bachelor thesis, which demonstrates the functionality described in the concept.

Ehrenwörtliche Erklärung

Ich erkläre hiermit ehrenwörtlich, dass ich die vorliegende Arbeit selbstständig angefertigt habe. Sämtliche aus fremden Quellen direkt oder indirekt übernommenen Gedanken sind als solche kenntlich gemacht.

Die Arbeit wurde bisher keiner anderen Prüfungsbehörde vorgelegt und noch nicht veröffentlicht.

Darmstadt, den 4. August 2011

Acknowledgement

I would first like to thank Dr. Martin Steinebach, and York Yannikos, for giving me the opportunity to write this thesis at the Fraunhofer SIT. Thank you both for all your support, guidance, and advice that kept me on the right path through the more challenging parts of this work.

I would also like to thank my amazing and lovely wife, Pari, who gave me unlimited patience, endured many movie nights without me, and worked tirelessly to lessen my burden. You gave me the confidence and encouragement I needed to get through this.

And to my wonderful mother Gisela, and father Michael, thank you for always supporting me, listening to me, and guiding me. The lessons you teach me are invaluable, and I am eternally grateful.

Table of Contents

Table of Figures	III
Tables	V
1 Introduction	1
1.1 Motivation	1
1.2 Project Description	2
1.3 Document Outline	2
2 Foundations and Security Requirements of Secure Logging	4
2.1 Terms Summarized.....	4
2.2 Security Requirements.....	5
2.3 Cryptographic Methods	6
2.3.1 Cryptographic Hash Functions	6
2.3.2 Hash Chains in Secure Logging.....	7
2.3.3 Symmetric Cryptography	9
2.3.4 Asymmetric Cryptography.....	10
3 State of the Art in Secure Logging Protocols.....	11
3.1 Architectural Considerations	11
3.2 Possible Log Attacks	12
3.3 Secure Logging Approaches.....	14
3.3.2 Schneier & Kelsey Approach	18
3.3.3 Ma & Tsudik's FssAgg Approaches	23
3.3.4 Holt's Logcrypt	26
3.3.5 Approaches not Evaluated in Detail	31
3.3.6 Summary.....	31
4 Requirements Analysis	33
4.1 Requirements.....	33
4.2 The Current Copying Process.....	34
4.3 Process Phases	35
4.4 Attacks and Boundaries	36
5 Concept	38
5.1 Processes and Architecture	38
5.1.1 Copying Process	39
5.1.2 Verification Process.....	40
5.1.3 System/Prototype Architecture	41
5.2 Secure Logging Principles.....	42
5.3 Signing Methods.....	42
5.4 Summary	42
6 Implementation.....	45
6.1 Development Environment	45
6.2 System Modules.....	46
6.3 Implementation Principles.....	47
6.4 Packages	49
6.5 Class Descriptions	50

6.6	JUnit Test Cases.....	52
6.7	Cryptographic Algorithms Used.....	52
6.8	Log Structure.....	52
6.9	Verification.....	54
6.10	Graphical User Interface.....	55
6.10.1	Alias Dialog.....	55
6.10.2	File Menu.....	56
6.10.3	Configuration Menu.....	57
6.10.4	Copy Tab.....	58
6.10.5	Log Integrity Tab.....	59
6.10.6	Data Integrity Tab.....	59
6.11	Prototype Setup.....	60
6.11.1	Key Database.....	60
6.11.2	Ini and Property Files.....	60
6.11.3	Prototype Tools.....	61
7	Performance Measurements.....	62
7.1	Performance Test Scenarios.....	62
7.2	Results.....	63
7.2.1	Copying.....	63
7.2.2	Verifying.....	66
8	Summary.....	68
8.1	Results Achieved.....	68
8.2	Prototype Boundaries.....	68
8.3	Outlook.....	68
9	Appendix.....	70
9.1	References.....	70
9.2	Performance Measurement Tables.....	71

Table of Figures

Figure 3.1: Remote Logging Principle	11
Figure 3.2: Asynchronous Remote Logging Principle.....	12
Figure 3.3: Secure Logging Focus Areas	14
Figure 3.4: Integrity of Log Data Protocols	15
Figure 3.5: FI-MAC Scheme	16
Figure 3.6: Schneier & Kelsey: Asynchronous Updates of Log File	19
Figure 3.7: Schneier & Kelsey: Generating Log Entries with Hash Chains	21
Figure 3.8: Schneier & Kelsey: Signing Each Hash Chain Value with a MAC.....	21
Figure 3.9: Ma & Tsudik: Verification Process	25
Figure 3.10: Holt's Logcrypt: Public Key Generation	27
Figure 3.11: Holt's Logcrypt: Signing Log Entries.....	28
Figure 3.12: Holt's Logcrypt: Verification Process.....	29
Figure 4.1: Omega Pro	35
Figure 4.2: Process Phases	35
Figure 5.1: New Omega Pro Copying Process	39
Figure 5.2: New Omega Pro Verification Process.....	40
Figure 5.3: System Architecture of the New Concept	41
Figure 6.1 System Components	46
Figure 6.2 Sample Copy/Verify Design Principle	48
Figure 6.3: Log Signer Factory.....	49
Figure 6.4 Package Structure	49
Figure 6.5: Log Entry Structure	52
Figure 6.6: Alias Dialog.....	56
Figure 6.7: File Menu.....	56
Figure 6.8: Open Configuration File Window	57
Figure 6.9: Configuration Menu	57
Figure 6.10: Log File Dialog.....	58
Figure 6.11: Copy Tab	58
Figure 6.12: Log Integrity Tab	59
Figure 6.13: Data Integrity Tab.....	59
Figure 7.1: Test PC Descriptions	62

Figure 7.2: Medion Copy Test Results	64
Figure 7.3: Dell Copy Test Results	64
Figure 7.4: Relation of Old and New Functionality	65
Figure 7.5: Medion Verification Test Results	66
Figure 7.6: Dell Verification Test Results	67

Tables

Table 2.1: Abbreviations for Secure Logging	5
Table 3.1: Attack Resistance of Approaches Analyzed.	32
Table 3.2: Special Properties of Presented Schemes.	32
Table 6.1: Cryptographic Algorithms Used	52

1 Introduction

This document has been written within the scope of the FORBILD project which has been established by the Fraunhofer SIT in Darmstadt, the Technical University of Darmstadt, and LSK Data Systems GmbH. The purpose of this project is to “assist the federal and state police in visual inspection of seized image data that potentially shows illegal content, namely child pornography. Privacy-preserving mechanisms are included in the project in order to reflect the delicacy of the image content”.

The inspection using the original data on the optical media is cumbersome and doesn't allow multiple people to analyze the data. Therefore, the data is copied to external discs. By analyzing the copied data the door is open for manipulating its content. This in turn, makes it more difficult to use the results as digital evidence in front of a court.

1.1 Motivation

In Kahn Consulting (1) securing log files plays an important role in using them as digital evidence:

“... computer log files are also increasingly used as evidence during the course of internal investigations, lawsuits, government investigations, audits, and other formal matters. As such, rather than viewing log file information as merely “technical” or tactical information, many organizations today view certain computer security log files as a unique form of “evidence” that must be managed in a manner that reflects its intended or possible future use.”

Kahn sees the fulfillment of the following challenges as a prerequisite in order to use digital information in a court of law:

“... Any organization wishing to rely upon electronic information for legal and regulatory purposes, or wishing to submit it as legal evidence must address two separate - but related - challenges. First, the information must be admissible – that is, it must be acceptable to the court or to the regulator.The second challenge for electronic information is that it must be credible. In other words, electronic information must be authentic, complete, and trustworthy enough to deserve to influence the outcome of a legal proceeding. Even if such evidence is found to be generally admissible, its integrity can nonetheless still be attacked, and it can be excluded or its influence on the proceeding can be severely diminished.”

Although today's practices are accepted by the courts (digital copies of media for analyzing data confiscated by the authorities can be used), a detailed analysis of the processes reveals that – if desired – the copied data can be easily modified, deleted, or amended with additional information. If misused, it could lead to drawing false conclusions in a case.

1.2 Project Description

One of the practical objectives of the FORBILD project is to propose a state of the art process that allows the integrity and authenticity of confiscated media to be upheld.

Unless specified differently in the rest of this document, the word “project” is specified by the scope described below.

Within the scope of this work, a state of the art software prototype supporting digital evidence for the LSK copying machine, called Omega Pro, has been developed. This machine helps the authorities become more productive when analyzing optical media, is widely used by the German authorities, and is admissible. However, it does not at all support authenticity and integrity.

Therefore, if digital evidence has to be proven in front of a court, using today’s mode of operation, this only can be achieved by going back to the original optical media. The authentication and integrity functionality implemented in this project’s prototype will save a lot of time by allowing one to check the copied data residing on hard drives, and using it in a legal proceeding.

In order to achieve the security requisites, a secure logging mechanism has to be integrated into the Omega Pro, documenting the copying process, and making tamper-attempts detectable. Note that, secure logging techniques do not prevent manipulation of the hard drive, but it certainly enables it to be detected.

In this document, various secure logging methods are examined to achieve the objectives mentioned above. Possible attacks are analyzed and matched against the requirements. Based on a thorough state of the art analysis of today’s secure logging techniques, a concept covering against most of the attacks is implemented. Performance tests conducted give an indication of the overhead generated by secure logging versus the current mode of operation. As mentioned above, the prototype cannot address all possible attacks. Limitations of and boundaries are described allowing to initiate organizational means to address the limitations.

1.3 Document Outline

This document is divided into the following sections: Chapter 2 gives a brief summary of terms used in this text and overviews security requirements of secure logging. Chapter 3 presents the state of the art in secure logging methods. In Chapter 4, the challenges of secure logging in context to using the Omega Pro system are examined and analyzed. Followed in Chapter 5 by, deriving a concept for the system based on the techniques evaluated in the state of the art. In Chapter 6, the prototype implementation of this concept on the basis of Java is described. Lastly, Chapter 7 compares the computational overhead of the new concept to the current system. In conclusion, Chapter 8



summarizes the document, showing that the requirements are fulfilled, and giving an outlook of further optimization.

2 Foundations and Security Requirements of Secure Logging

This chapter gives an overview of the most important terms and definitions used in this document in reference to secure logging. Furthermore, the terms integrity, authenticity and confidentiality used in the state of the art papers are defined. Lastly, the cryptographic methods used in this document are described. The main objective is to clearly define the terminology used to further describe the state of the art secure logging protocols, and to serve as a basis for reference.

2.1 Terms Summarized

What is a log file and how is it used in this document?

Log files record certain events, i.e. a system's activities. This information can be audited to analyze a system's behavior. In this document log files are denoted as L , containing log entries L_i which are appended to the file once they are created. The information stored in the i .th log entry is called an event, a log message, or log data, and is denoted as D_i .

What is secure logging and why is it important?

Simply put, secure logging is the process of creating a log file with properties which makes manipulation detectable. Its objective is to keep the integrity and authenticity of the log file. Without proof of these objectives, an auditor cannot rely on the authenticity/integrity of the file's content.

Log Machines

A log machine U receives the log data, puts it into a log entry, and appends it to the log file. This process is referred to in the rest of this text as the "logging process".

Trusted Server

A trusted server T serves as a repository for the log file and its associated metadata. The server is placed in a secure location.

Verifier

A verifier V is mainly a person whose intention is to audit the log. Before the audit is possible, a verification process is executed to ensure the log has not been tampered with.

Logging Architecture

The logging architecture is the environment for the process of secure logging. It usually consists of the logging machine and a trusted server.

Abbreviations

The following table further describes the roles of U , T and V described above. They are used in this document and other papers referred to (2); (3); (4).




Abbreviation	Security Status	Description	Picture
U	Untrusted	Logging machine (usually a server). This machine is considered untrusted, meaning that the probability for a successful compromise is high.	 U
T	Trusted	A trusted server, serving as the repository. It is assumed that T is resistant against attacks.	 T
V	Semi-trusted	A verifier, who checks the logs integrity in order to audit the log. This role is only trusted to read log records but not in any other way.	 V

Table 2.1: Abbreviations for Secure Logging

In general, it is assumed that connections between two arbitrary components (U , T or V) are secure, meaning that eavesdropping or other attacks do not have to be considered.

2.2 Security Requirements

The following defines the security requirements for secure logging. The terms below are used throughout this document and in the state of the art papers. The terms described below help in overcoming Kahn's second challenge (see 1.1) credibility of information, and are a prerequisite for proving digital evidence.

Integrity

Data is protected in terms of integrity when unauthorized modification can be proven. In the context of secure logging protocols, a definition of log file integrity is given by Rafael Accorsi (5):

A log file fulfills integrity when the following three requirements can be guaranteed:

- Accuracy: the entries have not been modified or reordered.
- Completeness: the log file, or some of its content, has not been deleted.
- Compactness: no content has been added to the file, e.g. by inserting forged records.

In summary, log integrity is assured when its records or structure cannot be modified without detection.

Authenticity

According to (6) authenticity is defined as the originality and credibility of a subject or object. Authenticity can be proven through a unique characteristic of the subject or object, e.g. a private key in which the subject or object is the only possessor. The process of proving someone's authenticity is called authentication.

Confidentiality

When important data, like a customer's bank account information or personal information, has to be transmitted via a public network, it is necessary the data remains unobtainable to other parties in the network. Confidentiality is achieved when it is impossible for unauthorized parties to access or read information. With regard to secure logging, Accorsi defines entry confidentiality in (5) when log data is not stored in plain text.

2.3 Cryptographic Methods

Secure logging techniques (which are presented in chapter 3.4) focus on proving a log's integrity. In order to do so, cryptographic methods are applied. This chapter is concentrates on cryptographic hash functions, hash chains, message authentication codes and digital signatures. More detail on these topics can be found in (7).

2.3.1 Cryptographic Hash Functions

The simplification purposes the following notation is used throughout this document.

For $x, y \in \{0,1\}^*$ the function $==$ is defined to be

$$x == y = \begin{cases} true, & \text{if } x = y \\ false, & \text{if } x \neq y \end{cases}$$

A cryptographic hash function denoted as: $H: \{0,1\}^* \rightarrow \{0,1\}^n$ has the ability to compress a string of arbitrary length into a string of length n-bits.

It has the following characteristics: the image $y = H(x)$ can be computed very efficiently, while it is "practically" impossible to compute the origin x of a given image y (called the hash). Practically means, that there is no efficient algorithm to compute the origin in a reasonable time. An example of such a function is the discrete logarithm explained in (7).

Cryptographic hash functions are not injective: $\exists x, y \in \{0,1\}^*: x \neq y \wedge H(x) = H(y)$. Elements mapped to the same image are called collision elements.

Based on the definitions above, cryptographic hash functions are categorized as weak, or strong collision-resistant. The following definitions are based on the definitions in (7):

- H is weak collision resistant if it is practically impossible to find a collision (x, x') based on a given x .
- H is strong collision resistant if it is practically impossible to find a collision at all.

In this document, it is assumed that cryptographic hash functions used in secure logging are strong collision resistant.

Hash functions are mainly used to check a message's integrity. When a message m is sent, its corresponding hash h is attached to it. The receiver of the message recalculates the hash and checks if both hashes match. In the following, this kind of verification denoted by:

$$verifyHash(m, h) := (h == H(m))$$

Widely used hashing algorithms are SHA-1 defined in (8), and MD5 defined in (9). According to Wang (10), the MD5 algorithm is not collision resistant anymore.

2.3.2 Hash Chains in Secure Logging

A log file is a file that is dynamic in the sense that on a continuous basis data is appended to it. This leads to the definition of dynamic data:

Let m_i ($i = 1, 2, \dots, n$, $n \in \mathbb{N}$) be data blocks (=generalization of a log entries). The concatenation

$$M_n := m_1 m_2 m_3 \dots m_n$$

of these data blocks is called dynamic data (of length n). The characteristic of dynamic data is that it changes its content throughout its lifetime by appending a new block of data.

Hash chains are defined as follows: let H be a strong collision-resistant hash function and M_n ($n \in \mathbb{N}$) dynamic data and C an arbitrary string. A hash chain K_n ($n \in \mathbb{N}$) is a recursive defined function hashing m_n and the previously generated hash K_{n-1} :

$$K_n := \begin{cases} C, & \text{if } n = 0 \\ H(K_{n-1} | m_n), & \text{if } n > 0 \end{cases}$$

The string C is called the initial seed of the chain.

In secure logging, hash chains are used for two purposes:

- Key generation
- Efficiently calculation hashes for dynamic data (using accumulative hash chains).

In order to distinguish between the two different hash chain purposes, accumulative hash chains will be denoted by A_n instead of K_n .

Key Generating Hash Chains

Hash chains can be used in secure logging processes requiring different encryption keys per log entry. Multiple keys are generated, which are all derived from one initial key.

By setting $C = K_0$ as the initial key and setting $M_n = \emptyset$ for all n in the definition above a hash chain is defined recursively:

$$K_n = H(K_{n-1})(n \in \mathbb{N})$$

Compared to symmetric encryption (Section 2.3.3) the advantage is: if party A wants to send data, encrypted by $K_n (n > 0)$ to party B there is no need to exchange this key with B. Sender A and receiver B must the initial key K_0 and the index n . This gives both parties the ability to compute K_n and therefore to encrypt or decrypt.

If hash chains are used for key generation, they have the ability to “forget” the previously used keys due to the property of a hash function (the origin cannot be computed). The pre-requisites for this are: the keys are deleted from the system immediately after they have been used, and the initial seed K_0 is held in a secure location.

Accumulative Hash Chains

The integrity of data can be verified by a normal hash function H as defined in chapter 2.3.1. If the data is dynamic, it is computationally more efficient to concatenate the current hash with the new block and then hash it, compared to hashing the entire message every time a new block of data is appended.

Let $M_n (n \in \mathbb{N})$ be dynamic data and m_i the i .th data block of M_n .

Due to the nature of hash functions

$$A_n = \begin{cases} C, & \text{if } n = 0 \\ H(A_{n-1}|m_n), & \text{if } n > 0 \end{cases}$$

guarantees the integrity of M_n .

If the receiver of (M_n, A_n) wants to check the integrity of M_n she computes a the hash chain value A_n^c and compares it with A_n . In case of $A_n^c == A_n$, M_n has been successfully verified. In the remainder of this document this is denoted by the Boolean function

$$\text{verifyHC}(M_n, A_n) := (A_n^c == A_n)$$

2.3.3 Symmetric Cryptography

Symmetric cryptography (also called secret key cryptography) can be used for many purposes such as transmitting data over an insecure channel, storing data on insecure media, or to perform an integrity check on data. In context of secure logging, the latter is the most important area which is the reason for a closer look up in the next subchapter.

Symmetric key algorithms, like the AES (11), use a secret key K_s negotiated by the communication partners for encryption as well as decryption. The benefit of such algorithms is that they are computationally inexpensive. On the other hand, there is a high key management effort if a communicating party needs to share a different secret key with each of its partners.

If encryption is used and data m needs to be en-/decrypted with the secret key K_s , symmetric en-/decryption is denoted as $E_{K_s}(m)$ and $D_{K_s}(m)$, respectively.

Message Authentication Code

Let K be an arbitrary set, H be the set of hash functions, then the family of hash functions MAC is defined to be:

$$MAC := \{h_{K_s} \in H \mid K_s \in K\} \subset H.$$

The message authentication code (MAC), also called Message Integrity Code (MIC), serves to retain the integrity and authenticity of a message among two or more communicating parties.

In the following the notation $MAC_{K_s} := h_{K_s}, (K_s \in K)$ is used.

The key K_s is shared between a sender, and receiver. As long as K_s is kept secret, the receiver of a message with an appended MAC can be certain of the authenticity and integrity of the message. Note that MACs are not used to undisputedly identify the originator of a message in the way digital signatures do (chapter 2.3.4). This is due to the fact that a symmetric keys needs to be shared among the communicating parties, making it impossible to identify the source of a message as either party can be the originator.

If z denotes the MAC created by the sender using K_s , the receiver of a message verifies z by applying the following verifying function:

$$verifyMAC(m, z, K_s) := (z == MAC_{K_s}(m)).$$

The MAC algorithm used in this work is the HMAC, introduced by Bellare et al in (12).

2.3.4 Asymmetric Cryptography

Asymmetric cryptography (also called public key cryptography) can be used for the same purposes as symmetric cryptography can. However, it provides stronger statements about a message's integrity and authenticity through the use of digital signatures. Digital Signatures, analogously to MACs, are important for secure logging techniques and are defined in the next subchapter.

Compared to symmetric key algorithms, asymmetric algorithms use two keys (pk, pub), denoted as private key and public key. Each of the communicating parties A and B own such a pair. The public key is registered at a trusted authority called Public Key Infrastructure (=PKI, see in (6)) and can be queried in form of X.509 certificates¹ by anyone to encrypt a message. If A wants to send a message to B she queries the public key of B from the PKI, encrypts the message and sends it to B. The receiver of the encrypted message uses his own private key to decrypt it. Analogously to symmetric algorithms, the en-/decrypted message is denoted as $E_{pk}(m)$ and $D_{pub}(m)$, respectively. Examples are the RSA or the ElGamal algorithm. Further information on this can be found in (7).

Digital Signatures

Compared to message authentication codes, digital signatures can be used to undisputedly authenticate the originator and content of a message.

Digital signatures make use of asymmetric algorithms and a PKI. If A wants to send a message to B with the intention of making sure that the message comes from her, A encrypts the message with her private key, attaches the encrypted message to the original one and sends both to B. The receiver B queries A's public key from the PKI, decrypts the attached signature and compares the result of the decryption with the message. In case of a match B knows that A must have sent this.

The following notations are used in this document to the signing and verification:

Let m be the message, (pk, pub) the key pair of A. Then

$$s := \text{sign}(m, pk) := E_{pk}(m)$$

defines the signature of the message using A's private key pk .

Based on the above the function

$$\text{verify}(m, s, pub) := (m == D_{pub}(s))$$

returns true if and only if using A's public key the decrypted signature s is identical to m .

Commonly used algorithms for the purposes described above are the RSA or the DSA. Further details with regard to these algorithms can be found in (7).

¹ <http://tools.ietf.org/html/rfc5280>

3 State of the Art in Secure Logging Protocols

The main focus of this chapter is on introducing state of the art secure logging approaches. Within this document the terms “schemes” and “techniques” are used as synonyms for the secure logging approaches. In order to examine the techniques, the architecture of a logging system and possible log attacks are defined. In the approaches presented, a security analysis maps the defined attacks into each approach with the intention of finding strengths and vulnerabilities. A summary of the findings concludes this chapter.

3.1 Architectural Considerations

It is important to determine in which environment the logging process is working. Log files are generated by various systems such as web servers, firewall devices, or network equipment. If the logging machine itself is considered to be trusted, meaning that intrusion is practically impossible, then no other resources have to be added. Creating a trusted machine can be realized by blocking it off from the rest of the world.

However, in the more likely scenario, the logging machine is considered as untrusted. Therefore, a trusted server is introduced serving as a repository for the log files. This remote logging technique is used to overcome the susceptibility of the untrusted servers.

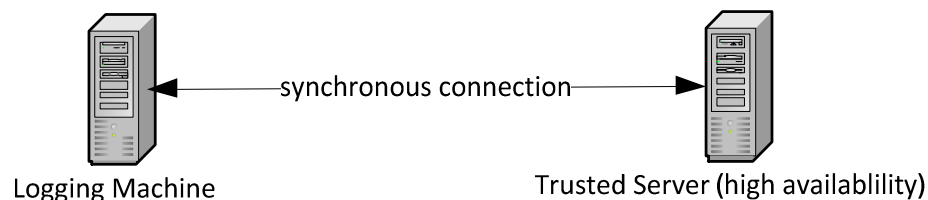


Figure 3.1: Remote Logging Principle

As shown in figure 3.1, the trusted server should be highly available in order to communicate, at any given time, with the logging machine.

A buffering mechanism is needed to cope with attacks (DoS attacks) against the trusted server which puts it into a state of being unable to continue to receive log entries. In that case, the logging machine temporarily stores the vulnerable log data in the buffer. An attacker who has control of the unsecure machine could possibly manipulate the log files via direct access or by copying manipulated log files to the untrusted server.

That being said, using an untrusted logging machine U and a trusted server T , the process of log generation and verification can be described as shown in Figure 3.2:

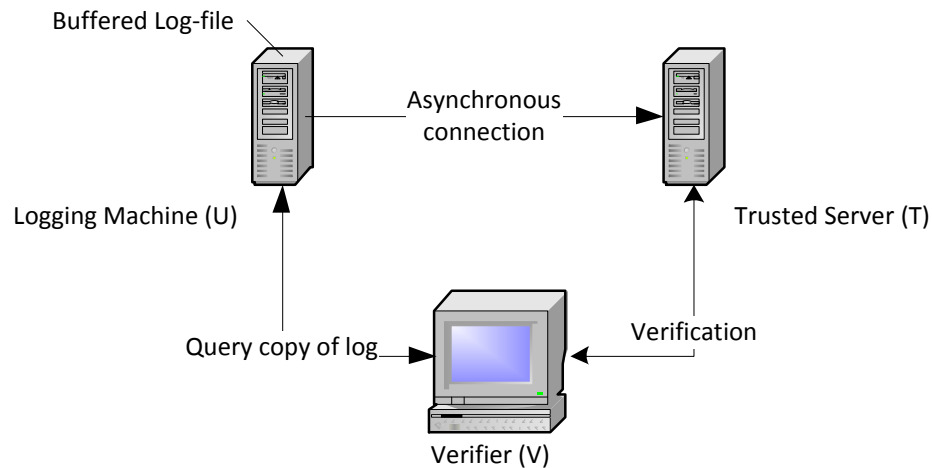


Figure 3.2: Asynchronous Remote Logging Principle

The untrusted server is the originator and retainer of the log file, updates to the log are sent asynchronously to the trusted repository.

When a verifier V has to audit the log, it retrieves it from the untrusted machine. In order to ensure the log file has not been changed spuriously, the verifier executes a verification process with the repository, checking for the log's integrity and authenticity.

Current secure logging approaches use this scenario as a basis and deliver mechanisms to cope with the possible attacks on log files.

3.2 Possible Log Attacks

This chapter presents possible attacks on logging systems which are analyzed within this document. These attacks have been created in reference to this project only, and are not defined in any other work.

Using the dynamic data definition for the log $L = L_1L_2L_3 \dots L_n$ the adversary's possibilities with regard to attacks on L are explained. Regardless of the intruder's objective, the attacks listed below could be executed from within or outside of the logging machine, during the logging process, or after it has terminated:

Modifying Attack

The intruder creates a false block of log entries $L'_i \dots L'_j$ ($1 \leq i \leq j \leq n$) and replaces the original log entries by the false entries. The resulting log would be: $L' = L_1 \dots L_{i-1}L'_i \dots L'_jL_{j+1} \dots L_n$.

Insertion Attack

The intruder inserts false entries. In example, a block of false log entries $L'_1 \dots L'_{1+m}$ ($m \in \mathbb{N}$) is inserted after the i .th entry of the correct log. The resulting log would be: $L' = L_1 \dots L_j L'_1 \dots L'_{1+m} L_{i+1} \dots L_n$.

Reordering Attack

The intruder switches the position of the entries, which in turn changes the row of events. This can be considered as a sequence of modify and insertion attacks. Therefore reordering attacks are not analyzed.

Deletion Attack

The intruder deletes current log entries. In example, a series of log entries $L_i \dots L_j$ ($1 \leq i \leq j < n$) is deleted from the beginning or the middle of the log. The resulting log would be: $L' = L_1 \dots L_{i-1} L_{j+1} \dots L_n$.

Truncation Attack

A block of log entries $L_{i+1} \dots L_n$ ($1 \leq i < n$) is truncated from the end of L . The resulting log would be: $L' = L_1 \dots L_i$.

For further analysis, it needs to be distinguished if the attacker decides to continue or stop the logging process after this attack. Truncation attacks can be detected if the logging machine continues to log. In this case, the truncation is treated equivalent to a deletion attack. Therefore, further analysis assumes that the logging process stops.

Total Deletion Attack

The intruder removes the entire file from the system.

Appending Attack

A block of false log entries $L'_1 \dots L'_{1+m}$ ($m \in \mathbb{N}$) is appended to the existing log. The resulting log would be: $L' = L_1 L_2 L_3 \dots L_n L'_1 \dots L'_{1+m}$.

Verification Attack

Some secure logging approaches use symmetric cryptography to protect a log's integrity. Thus, the same key is needed for the logging and verification process. If the verifier is an attacker, she is able to forge a log, remaining undetected, unless the log is compared to the original file at some point.

Delayed Detection Attack

The Delayed Detection Attack was defined by Di Ma and Gene Tsudik (3) and is described in chapter 3.3.6.

3.3 Secure Logging Approaches

This section examines the secure logging state of the art approaches. Figure 3.3 gives an overview of the main focus areas of secure logging. The hierarchical structure shown in this figure is used to filter out areas that are not relevant to the logging concept that has been implemented in this work.

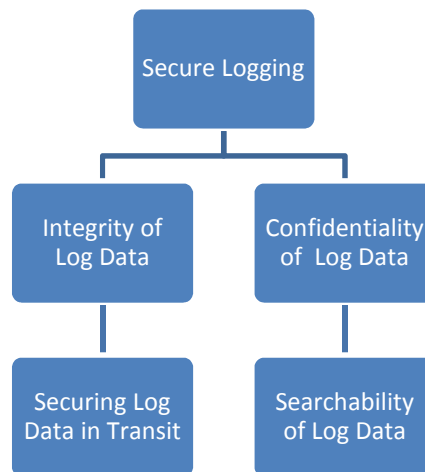


Figure 3.3: Secure Logging Focus Areas

The state of the art approaches deal with two main areas of secure logging: integrity and confidentiality.

Confidentiality of Log Data plays an important for logs containing sensitive data. These logs have to be protected from unauthorized access. Therefore, encryption is used to hide the data contained in these logs. Whenever these logs have to be searched by authorized people performance impacts have to be considered.

Searchability of Log Data is achieved by implementing keywords, indexes and mechanisms allowing to quickly identify information that the searcher is allowed to see.

Integrity of Log Data deals with secure logging on the logging machine (untrusted or trusted) generating the log.

Securing Log Data in Transit deals with securing the messages during the transmission phase between two servers.

The main scope of this document is limited to the focus area integrity of log data. This is based on the following:

- The log files generated during the copying process of the Omega Pro do not contain sensitive information and do not need to be encrypted.

- The log files generated are small and there is no need to search.

However, in order to be thorough, these topics will briefly be touched upon in section 3.3.5.

Figure 3.4 gives an overview of secure logging protocols dealing with integrity of log data. These approaches are presented in the remainder of this chapter.

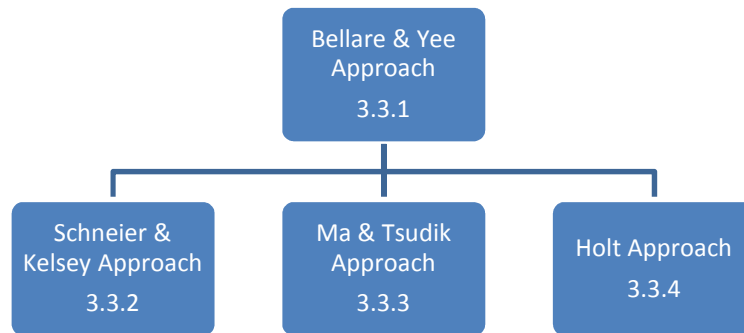


Figure 3.4: Integrity of Log Data Protocols

The **Bellare and Yee approach** defined in (4), forms the basis of all other approaches by introducing new ways of signing logs, resulting in the integrity of the log data being guaranteed up to the point of intrusion.

The **Schneier & Kelsey approach** defined in (2) refines the Bellare and Yee approach in the sense that detailed protocols describe the communication between U, T and V , and the logging process.

The **Ma & Tsudik approach** defined in (3) is driven by the desire to eliminate the truncation attack weaknesses of the Schneier & Kelsey approach detected by both authors. Their approach is based on the Bellare and Yee principles as well and is truncation attack resistant.

The **Holt approach** defined in (13) introduces asymmetric cryptography to allow public verification of log file data.

Further approaches used in secure logging enhance the approaches described above. Due to the requirements mentioned in chapter 4 they were considered out of scope.

3.3.1 Bellare & Yee's Forward Integrity MAC-Scheme

In (4) Bellare and Yee defined a scheme that helps detect alteration in log files up to an intrusion point. This scheme is the basis for further secure logging schemes described in this document.

The following defines forward integrity and describes Bellare and Yee's FI-MAC Scheme.

Forward Integrity

Forward integrity (FI), also called forward security assures that created MAC/digital signatures generated prior to an attack remain valid and unforgeable, even in the case of a key-compromise.

This is realized by frequently changing the signing key and erasing the previously used one, giving an adversary no possibility to gain access to keys used in the past. FI implies integrity as defined in chapter 2.2.

The FI-MAC Scheme

In Bellare and Yee's FI-MAC Scheme, Forward Integrity is achieved by using a hash chain for key generation. Time is split into epochs $E_i := \{t: T_i \leq t < T_{i+1}\}$ which define the life cycle or "durability" of an authentication key K_i . Once the epoch is over, a new corresponding key $K_{i+1} = H(K_i)$ is computed. In order to assure that no attacker is able to regenerate any of these keys, the initial key of this chain has to be kept in a secure location (in this document T) and old keys are erased immediately after the new key has been generated.

Let $L = L_1L_2L_3 \dots L_{i-1}$ be the current log on U . Figure 3.5 shows how these keys are used to build MACs (denoted as z_i) for the i .th and $i + 1$.th incoming log records. The life cycle of an authentication key expires every time a log entry has been signed.

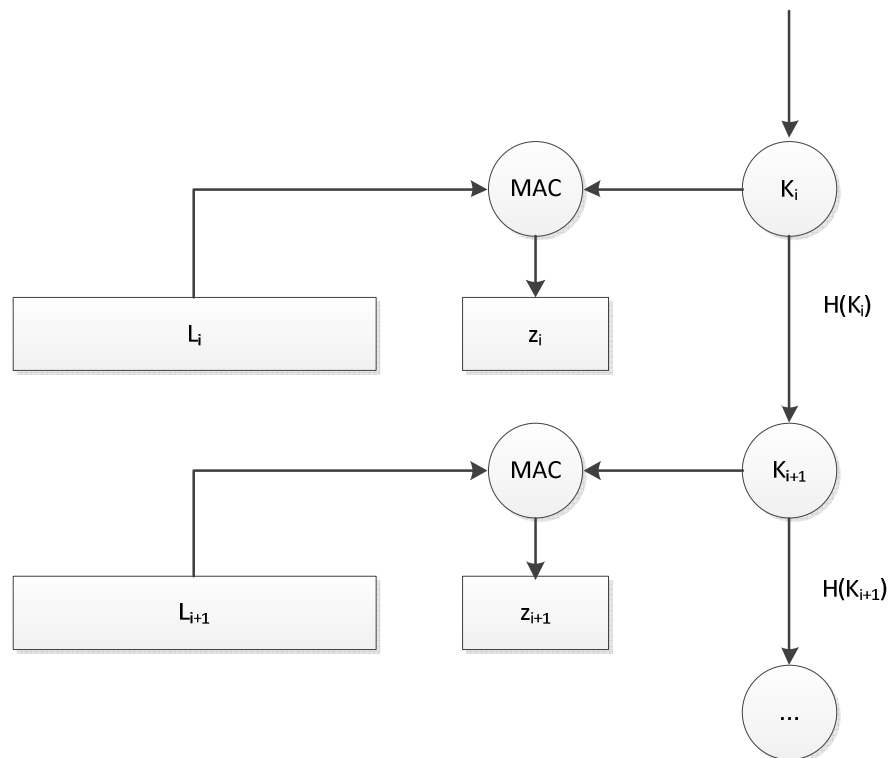


Figure 3.5: FI-MAC Scheme

Let $L'_i := (L_i, z_i), (1 \leq i \leq n \in \mathbb{N})$ be the i .th concatenated log entry. The resulting log is defined as $L' = L'_1 L'_2 L'_3 \dots L'_n$. Note that through hashing the authentication key, all keys differ from each other. This way, every log entry is assigned by one indisputable key. In addition, K_0 is not used to sign any entries, this is chosen due to the fact that a log starts at index 1 and thus it makes the denotations better readable.

An attacker who breaks into the system at time $t_c \in E_i$, might be able to gain possession of the current key K_i , but she will not find any signing key from the past, since they have already been removed.

Verification

With the knowledge of the hash function H and its corresponding initial key K_0 , the FI-MAC verification is achieved by checking each MAC:

$$L' \text{ valid} \leftrightarrow \forall j \in \{1, \dots, n\}: \text{verifyMAC}(L_j, z_j, K_j) = \text{true}$$

Security Analysis

At the moment of intrusion, let $L' = L'_1 L'_2 L'_3 \dots L'_n$ be the log with its corresponding MACs, K_0 the seed of the hash chain used for key generation and K_{n+1} the current authentication key.

It is assumed that the attacker takes over the entire system, gains control of the hash chain function H , as well as the current signing key K_{n+1} . Note that the intruder has no access to the seed K_0 since it is safely stored on T .

The attacker cannot apply any of the following attacks on pre-compromised log data:

- (1) **Insertion or Modifying Attack:** the intruder cannot insert or modify a forged log entry L'_i because she is not in possession of K_0 and therefore is not able to compute K_i to build a valid MAC of this entry.
- (2) **Deletion Attack:** Each authentication key is assigned to an explicit log entry. Therefore, the verification procedure would fail as soon as log entries (and MACs) are missing. For example, it is assumed that the adversary deletes the second entry and MAC from L' . The resulting log would be $L'' = L'_1 L'_3 \dots L'_n$. Due to this action during verification the second key K_2 would be assigned on log record L_3 . Therefore verification would fail: $z_3 \neq \text{MAC}_{K_2}(L_3)$.
- (3) **Total Deletion Attack:** since T knows about the seed K_0 , the trusted server acts as a witness for the existence of the log. For this reason, deleting the entire file destroys evidence but this deletion would be detected.

Besides the fact that this scheme provides forward integrity, it is also effective in exposing which entries have been tampered with and which entries remained valid due to the verification process. Though,

this is valid as long the attacker has not inserted or deleted any entries, which would destroy the assignment between authentication keys and log entries (see deletion attack above).

The scheme, as it is presented in this document, does not prevent the log from:

- (4) **Truncation Attacks:** If the server does not continue to log, an attacker can mount a truncation attack without the verifier being able to detect it. This is because the verifier does not know how many log records have been created by the logging machine.
- (5) **Verification Attacks:** Every verifier must have access to the initial authentication key K_0 . Thus a malicious verifier gains the ability to forge the entire log and therefore, the log's authenticity cannot be guaranteed.
- (6) **Appending Attacks:** At the point of intrusion, the attacker gains possession of the current signing key K_{n+1} . This enables her to append an arbitrary amount of entries without detection.

Conclusions drawn from Bellare & Yee's approach

Logging approaches, explained in the next chapters are derived from the FI-MAC scheme. Thus, the following assumptions and definitions hold for the remainder of this document:

- Forward Integrity Protection: Each approach implementing forward integrity is protected against the attacks as mentioned in this chapter.
- Initial Key Commitment: any initial authentication key K_0 , used for the FI purpose is transmitted from the logging machine U to the trusted server T at the beginning of the protocol. As soon as the key is committed, it is evolved and erased from the system; it is not used to sign a log entry.
- It is assumed that U is not compromised before the initial key has been transmitted to T .
- Forward Integrity Technique: After a key has been used to create a MAC (or signature), it is assumed that this key is immediately evolved and erased securely from the system.
- Authenticity can only be guaranteed when it is impossible to mount a verification attack.
- A MAC signing a log entry is called an FI-MAC.

3.3.2 Schneier & Kelsey Approach

In (2) Schneier and Kelsey introduced a protocol targeting forward integrity, authenticity, and entry confidentiality.

Schneier and Kelsey’s protocol provides a detailed description of the logging system’s architecture, how interactions between the system’s components take place, and how each log entry is built. The protocol is divided into five subprocesses:

- **Log File Initialization** – a key exchange protocol describing the commitment of the initial key K_0 .
- **Log File Close Down** – the file is closed with a special entry, implying that appending log entries is impossible afterwards.
- **Log File Validation** – the log is validated by T for its integrity and authenticity after it has been closed.
- **Log Entry Creation** – log entries are appended to the file, following the forward integrity technique.
- **Log File Verification** – a semi-trusted verifier V queries the file and executes a verification process together with T .

All processes mentioned above are briefly described in the following sections. The explanations in this document represent the principles of the Schneier and Kelsey approach. Detailed descriptions are either omitted or only partly mentioned. More phase detail can be found in (2).

Communication Processes

The first three subprocesses concentrate on the communication between the logging machine and the trusted server. Figure 3.6 describes the general interaction between the logging machine U and the trusted server T .

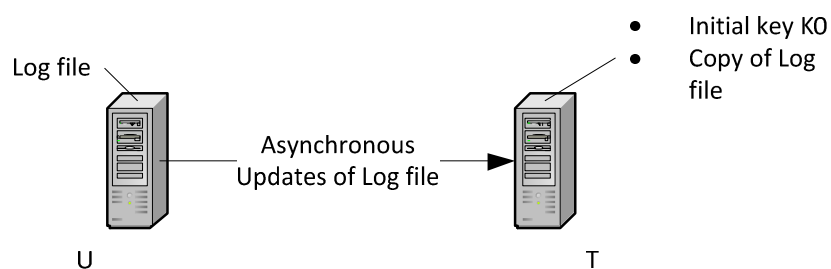


Figure 3.6: Schneier & Kelsey: Asynchronous Updates of Log File

The log file resides on both servers. U houses the original log file, and is responsible for processing log data by creating an entry and adding it to the file. Whenever a connection is available, U updates T by transferring the most recent log records.

Log file initialization

During the log file initialization the communication between U and T results in the agreement of the initial hash chain key K_0 between the unsecure Server U and the trusted server T . In their paper, Schneier and Kelsey detail this communication process using asymmetric cryptographic methods to authenticate, and a challenge response procedure to commit K_0 .

Log file close down

Let L_n be the last log entry to be logged. U still has to complete three final steps:

1. Writing a final log record L_{n+1} , declaring that the file finishes at this point.
2. Securely erasing K_{n+1} .
3. Closing the file.

With the next available connection, the closed file is then transmitted to T .

Log file validation

T knows K_0 and the closed log $L = L_1L_2L_3 \dots L_{n+1}$. Therefore, T is able to verify the content of the log file.

Logging Technique

Unlike other approaches, Schneier and Kelsey define certain components a log entry should contain. In total, four components build one log entry L_i . The first two are used to achieve confidentiality, while the other two are used to prove integrity. The following will briefly introduce these components:

- W_i : A permission mask. Every verifier V (or a group of verifiers) authorized to read the entry L_i is mentioned in this field. If a verifier is not mentioned in this component, access will be denied.
- $E_{F_i}(D_i)$: In order to make log data D_i unreadable to unauthorized parties, a symmetric cryptography algorithm E is used to encrypt the data with a symmetric key F_i . This key is derived from the authentication key K_i and therefore differs with every entry.
- A_i : An accumulative hash chain for the dynamic log file.
- z_i : Analogously to the FI-MAC scheme, this is the MAC of the i .th log entry.

The components mentioned above supporting confidentiality are not explained any further in this document. This is due to the requirements mentioned in chapter 4. Therefore, W_i and $E_{F_i}(D_i)$ are summarized as x_i .

Once the initial key K_0 has successfully been exchanged with the trusted server, U is allowed to start adding entries. For this, let $L = L_1 L_2 L_3 \dots L_{i-1}$ be the current log, K_i the current authentication key and x_i the summarized data to be logged. U creates the i .th entry L_i as follows:

Step 1: Recall the definition of an accumulative hash chain (chapter 2.3.2). In Schneier and Kelsey the hash chain is defined as follows:

$$A_i = \begin{cases} C = 0, & \text{if } i = 0 \\ H(A_{i-1} | x_i), & \text{if } i > 0 \end{cases}$$

Each accumulative hash chain value is a representative hash value for the newly generated dynamic X -values. This way, a representative hash value for $X_i = x_1 x_2 \dots x_i$ is given by A_i and added to the log entry (see Figure 3.7).

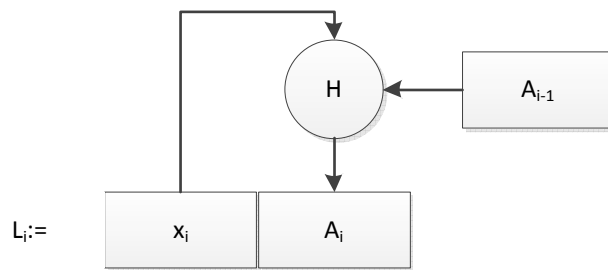


Figure 3.7: Schneier & Kelsey: Generating Log Entries with Hash Chains

The initial seed for this chain is defined as $A_0 = 0$ and will be initiated after the initial key commitment has been taken place.

Step 2: Figure 3.8 illustrates the employment of the Bellare and Yee approach by building the MAC over the accumulative hash chain value using the evolving key K_i : $z_i := MAC_{K_i}(A_i)$

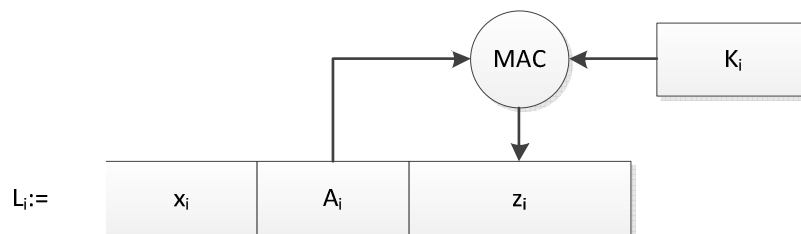


Figure 3.8: Schneier & Kelsey: Signing Each Hash Chain Value with a MAC

Verification

Let $L = L_1L_2L_3 \dots L_i$ be the current log on U while each entry is created as described in the previous chapter.

Based on the way the log entries are constructed, the following theorem implies a successful verification of the log:

$$\text{verifyMAC}(A_i, z_i, K_i) = \text{true} \rightarrow \forall j < i: \text{verifyMAC}(A_j, z_j, K_j) = \text{true}$$

For this reason, it is only necessary to validate the last log entry L_i .

A verifier V is allowed to audit a log file at any time, the log does not necessarily have to be closed. In order to verify the log it is not required that L has been updated to T yet. When V queries the log file from U , it is not in possession of any authentication key. Therefore she is not able to read log entries or verify the MACs for their correctness and has to consult the trusted server to aid in this process. Though, the verifier can check if $\text{verifyHC}(L, A_i)$ returns true or not. Only a valid hash chain value A_i guarantees that the corresponding MAC can be verified. If the check fails, the verifier does not need to consult T . Under the assumptions that $\text{verifyHC}(L, A_i) = \text{true}$, V transfers the following information to T : i, A_i, z_i .

With the received data from V , the trusted server can prove the validity of the log by applying the verify function: $\text{verifyMAC}(A_i, z_i, K_i)$. According to the theorem above, if this function returns true, it proves that all other entries ($j < i$) must be valid as well.

Security Analysis

By applying the FI-MAC scheme Schneier and Kelsey's approach automatically guards against all attacks the FI-MAC scheme covers. Additionally, the scheme detects the following attack:

- (1) **Verification Attacks:** Due to the fact that T is considered as trusted and that K_0 stays on T during every phase of the protocol, it is impossible that K_0 falls into an adversary's hands. Therefore, authenticity can be guaranteed.

Under certain circumstances, the scheme detects the truncation and appending attack. For this reason the attacks are considered as partly detectable:

- (2) **Truncation Attacks:** Uploading the newest log entries to the trusted server guarantees that there is a second instance witnessing that these entries actually exist. If an intruder truncates entries which have already been uploaded to T it will be detected by means of verification.

This statement is not true for log entries that have not been transmitted to T yet. The explanation is consistent with the one from Bellare and Yee: T does not know the true index of the current log.

(3) **Appending Attack:** An attacker cannot execute this attack if the log file has been closed. Unless the log file has been closed, an intruder can gain possession of the current FI-MAC keys and therefore append new entries to the log. For this reason, the authors request that a system be able to detect such intrusions and immediately log them. This way, an intrusion would be indicated and point to the log entries from where they are invalid.

Di Ma et al. (3) analyzed the protocol assuming that the current log has not been updated to T yet. In this case, they discovered that this approach is vulnerable against the delayed detection attack:

(4) **Delayed Detection Attacks.** For this, let $L = L_1L_2 \dots L_i \dots L_n$ be the current log, while $L_{i+1} \dots L_n$ has not been uploaded yet. The intruder gains access of the current signing key K_{n+1} .

An attacker modifies non-uploaded log records, but leaves the z_j -values ($j > i$) unmodified. She then corrects the hash chain A_j for every entry from where modification took place (note that this is possible because the attacker knows the hashfunction). With the gained current signing key K_{n+1} , she adds at least one forged entry with a generated A_{n+1} and z_{n+1} .

A verifier receiving this log will approve the integrity check and forward $(n + 1, A_{n+1}, z_{n+1})$ to T . Due to the fact that $z_{n+1} = MAC_{K_{n+1}}(A_{n+1})$ is valid and that T has not been updated yet, the forge will not be detected immediately. T can only detect the fraud in a delayed state as soon as it receives an update from the compromised U .

3.3.3 Ma & Tsudik's FssAgg Approaches

By analyzing the Schneier and Kelsey protocol, Di Ma and Gene Tsudik discovered the vulnerability of the approach (=scheme) to truncation and delayed detection attacks (3). To overcome these flaws they introduced the Forward Secure Sequential Aggregate (FssAgg) schemes in (14) and (3)).

These schemes have been developed on the basis of symmetric (FI-MAC) and asymmetric signing procedures. Being that, both schemes are based on the same principles and asymmetric cryptography is computationally more expensive, this document does not cover the asymmetric scheme. For more detail on the omitted approach please see (3).

Logging Technique

Recall the definition of an accumulative hash chain (chapter 2.3.2):

$$A_i = \begin{cases} C, & \text{if } i = 0 \\ H(A_{i-1} | m_i), & \text{if } i > 0 \end{cases}$$

Where m_i denotes the i .th data block of the dynamic data M_n .

In (3) accumulative hash chains are employed together with the FI-MAC scheme. Each m_i is replaced

by a FI-MAC $z_i := MAC_{K_i}(L_i)$ in the definition above. The authors define Forward Secure Sequential Aggregate (FssAgg) tag to be:

$$A_i = \begin{cases} \emptyset, & \text{if } i = 0 \\ H(A_{i-1} | z_i), & \text{if } i > 0 \end{cases}$$

This implies the this definition is the definition of hash chains applied to the dynamic data $Z_n := z_1 z_2 z_3 \dots z_n$. Therefore a tag is nothing more than a hash chain element.

Compared to the schemes which have been presented so far, this construct accumulates all signatures into one tag. Thus, it is not necessary anymore to attach them to their corresponding log record.

Ma et al. define two FssAgg tags. One tag is assigned for the verifier (denoted as $A_{V,i}$) while the other one is assigned for the trusted server (denoted as $A_{T,i}$). Both values differ from each other because of the use of different key generating hash chains, initiated under unequal seeds $K_{V,0}$ and $K_{T,0}$. Following the principle applied for the FI-MAC scheme, these keys are committed to T at the beginning of the protocol.

When a new log entry L_i appears to be signed, both tags are updated with the corresponding MACs from the logging machine respectively.

Log close down

As in the Schneier and Kelsey approach, U closes the file with a final “close” entry L_{n+1} . It then removes the last signing keys and sends $L = L_1 L_2 L_3 \dots L_{n+1}$ and $A_{T,n+1}$ to T . T then validates the log as described in the verification process (see next Section) with its own initial key $K_{T,0}$.

Verification

Figure 3.9 shows V verifying the message with U and T . First, it retrieves the current log with the FssAgg tag, which was generated for V , from the logging machine (together denoted as $(L, A_{V,n})$). In order to verify the log, it needs the corresponding authentication key $K_{V,0}$, which is sent by T .

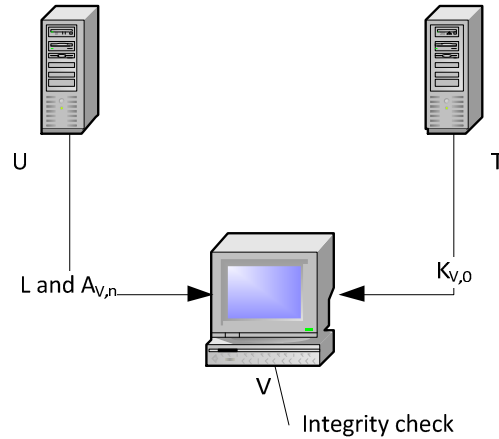


Figure 3.9: Ma & Tsudik: Verification Process

The verifier runs the following verification steps:

- With $K_{V,0}$, she builds the MACs analogously as described in Bellare and Yee's approach (see chapter 3.3.1). Each z_i is part of the dynamic data $Z_n := z_1 z_2 z_3 \dots z_n$
- If all MACs are created V can verify the FssAgg tag $A_{V,n}$ with the following theorem:

$$L \text{ valid} \leftrightarrow \text{verifyHC}(Z_n, A_{n,V}) = \text{true}$$

Security Analysis

Ma and Tsudik found the Schneier and Kelsey approach suffering from delayed detection and truncation attacks. Both of these attacks are caused due to the fact that the trusted server has not been updated synchronously. A FssAgg tag is used to summarize and build an representative value of the current log status which is updated right away. Thus, these attacks are not an issue anymore:

Let $L = L_1 L_2 L_3 \dots L_i$ be the current log status, $A_{V,i}$ respective $A_{T,i}$ the current FssAgg tags on U and $K_{V,i+1}, K_{T,i+1}$ the current signing keys, respectively. An attacker cannot execute the following attack:

- (1) **Truncation Attack:** An adversary aiming to truncate a block of entries $L_{j+1} \dots L_i$ ($0 < j < i$) also has to forge the two FssAgg tags $A_{V,i}, A_{T,i}$, otherwise verification will fail. Because the attacker owns neither $K_{V,j+1}$ nor $K_{T,j+1}$, she cannot execute this attack. Therefore, the truncation of a log is not possible without detection.

The following attacks are partly detectable:

- (2) **Verification Attack:** By handing $K_{V,0}$ out to the (malicious) verifier, it gives her the opportunity to forge entries and the tag. Afterwards, she could publish this log to the other verifiers, claiming it as the original. However, if the verifiers decide to consult the trusted

server to validate the forged log, the attack can be detected. T has its own secret key. Therefore, it can authenticate the log.

(3) **Appending Attack:** Due to the same reasons explained in the Schneier and Kelsey approach appending entries is not possible anymore after the file has been closed.

If the process is still running, both current authentication keys are stored on the logging machine, which gives an intruder total power over the log. The current authentication keys can be used to fake the log. Appended entries cannot be distinguished from real ones.

3.3.4 Holt's Logcrypt

Jason E. Holt introduced his approach under the name "Logcrypt" (13). Its main focus is the achievement of forward integrity using public key cryptography.

Holt extended the idea of Bellare and Yee by replacing each MAC with a digital signature:

"The primary disadvantage of the symmetric system just described is that verification of a MAC requires the same key that was used to create it. This means that anyone with the ability to verify a particular log would also appear correct.

Public key cryptography provides the ability to separate signing from verification and encryption from decryption."

This idea overcomes the problem of building a MAC and having it verified by V via the same symmetric key. Schneier and Kelsey tried to solve this problem by using T as the only holder of this key. Consequently, verification could only take place on T , costing computational resources and requiring a two server infrastructure. In Logcrypt on the other hand, every verifier V is able to check the log, relieving T from its workload. It is no longer significant, whether V can be trusted or not, since entry forgery is not possible through the use of public keys. As a result, this scheme is useful in logging environments where the logging mechanism has to be publicly verifiable.

Holt presents a symmetric approach using the FI-MAC scheme from Bellare and Yee as well. Due to the fact that both approaches barely differ from each other, a description of Holt's symmetric approach is omitted in this document. More detail can be found in (13).

The following concentrates on the description of the publicly verifiable approach.

Logging Technique

Comparable to secure logging approaches based on symmetric cryptography, U generates an initial, asymmetric key pair (pk_0, pub_0) at the beginning. The private key pk_0 is used to sign the first log entry while the public one serves to authenticate the log during verification. For this reason, pub_0 needs to be

stored in a secure location, accessible to verifiers. In his work, Holt does not declare a specific location for this key. Therefore, T is used in this explanation.

Besides the standard incoming log entries, Holt introduces an additional type of log entry holding public keys to verify entries which were signed during the signing process. In this document, this list of keys is denoted as PKL (public key list). The following shows how this entry is employed into the logging process:

Step 1: U generates $n + 1$ ($n \in \mathbb{N}$) key pairs $((pk_1, pub_1), (pk_2, pub_2), \dots, (pk_{n+1}, pub_{n+1}))$

Step 2: The first n public keys are extracted into a list $PKL_0 = (pub_1, pub_2, \dots, pub_{n+1})$ which builds the first log entry $L_0 := PKL_0$.

Step 3: In order to guarantee authenticity and integrity for this list, the initial private key pk_0 is used to sign this entry (see Figure 3.10). The signature is denoted as s_0 .

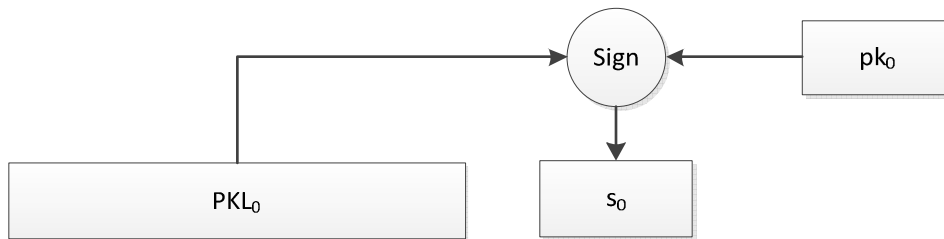


Figure 3.10: Holt's Logcrypt: Public Key Generation

Step 4: The logging machine now has $n + 1$ private keys to sign future entries. Figure 3.11 shows how the first n private keys are used to sign incoming log records $L_1 \dots L_n$.

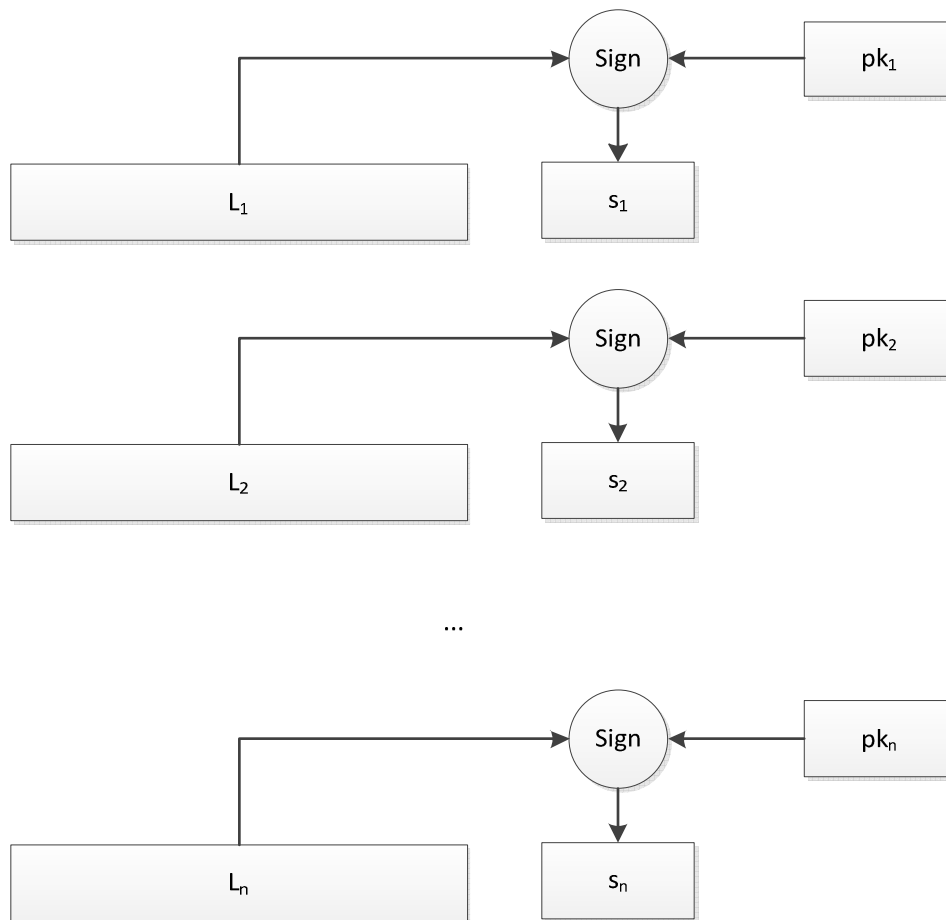


Figure 3.11: Holt's Logcrypt: Signing Log Entries

Note that the order of key-assignments is important. Each key is assigned sequentially as it is listed in PKL_0 . If the order is unknown, corresponding public keys cannot be associated to the log entries anymore.

Step 5: Once all n keys are exhausted, the $n + 1$.th private key is needed to repeat the entire procedure with a new list of $n + 1$ generated key pairs, beginning analogously from step 1.

Steps 1 to 5 build the asymmetric method of creating a log. Note that this scheme is presented from the beginning of the protocol to show the importance of the initial key pair. In general, the scheme can be repeated as long as desired.

Verification

Let n be defined as in the previous section. In order to verify the log file, V queries pub_0 from the trusted server. The verification process shown in Figure 3.12 shows how a successful verification of the first entry allows a verification of the next n entries,

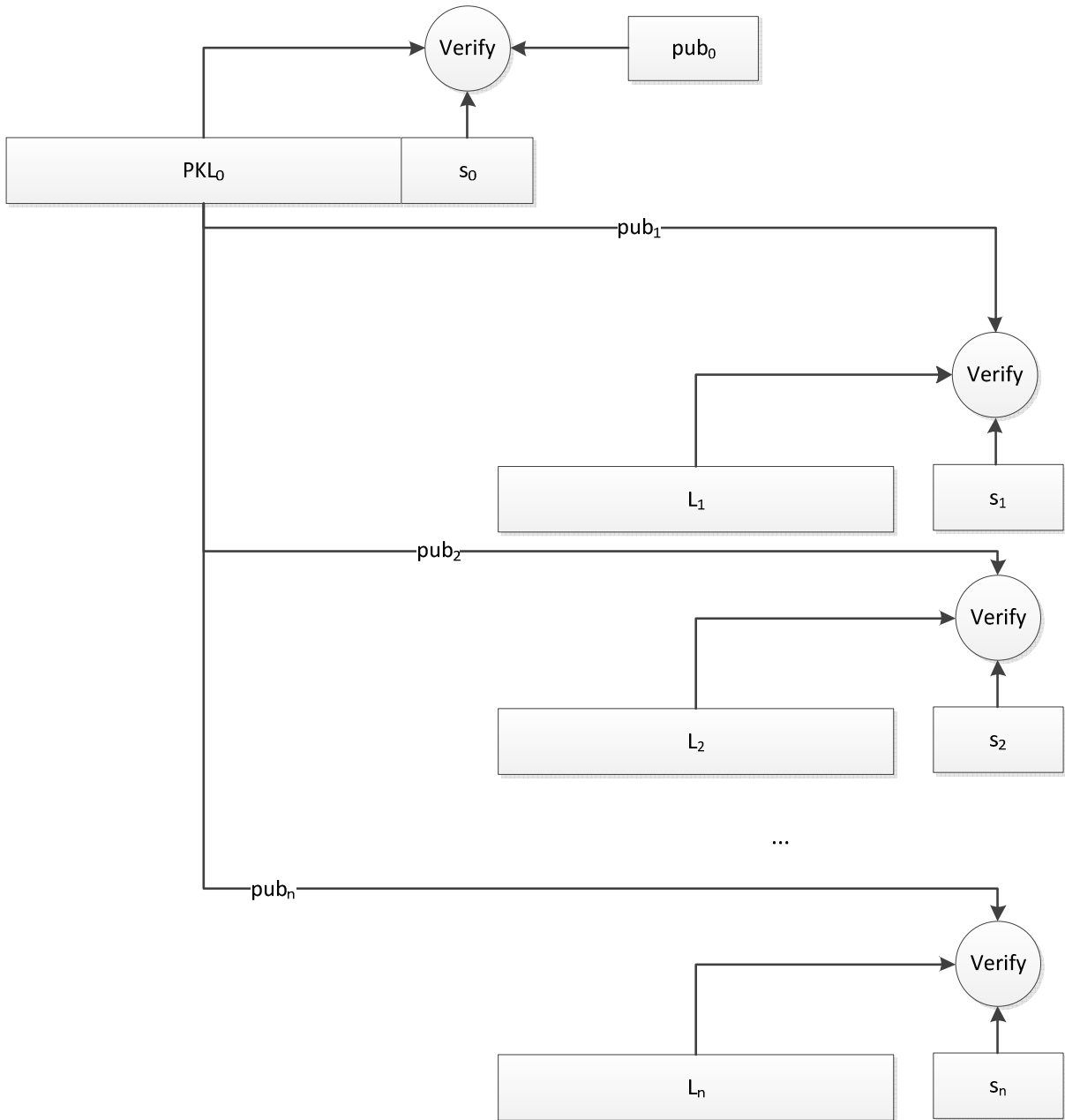


Figure 3.12: Holt's Logcrypt: Verification Process

Let $L = L_1L_2 \dots L_n$ be a log file, $0 < i < n + 1$, $s_i := \text{sign}(L_i, pk_i)$, $L'_i := (L_i, s_i)$, $L' := L'_1L'_2 \dots L'_n$ and $s_0 := \text{sign}(PKL_0, pk_0)$. Because T is trusted, the verifier can be sure about the correctness of pub_0 .

If

$$\text{verify}(PKL_0, s_0, pub_0) = \text{true}$$

PKL_0 is authentic, which implies that the next $n + 1$ public keys are authentic. This means, that the next n signatures can only be valid if and only if

$$\forall j \in \{1 \dots n\}: \text{verify}(L_j, s_j, \text{pub}_j) = \text{true}$$

is proven. Consequently, if one of the signatures is invalid, the verification process will fail.

Security analysis

Because Holt's Logcrypt provides forward integrity with public key cryptography, all attacks listed in chapter 3.3.1 are covered by this approach as well.

Logcrypt does not guard against truncation or appending attacks:

- (1) **Truncation Attack:** Unless the number of log entries and corresponding signatures is known by a second trusted instance, this approach is vulnerable to this attack.
- (2) **Appending Attack:** Due to the fact that the attacker gains possession of the current signing keys in this scheme, she can append an arbitrary amount of log entries to their corresponding signatures.

3.3.5 Approaches not Evaluated in Detail

All approaches mentioned in this chapter have been filtered out being that they are not integral to the requirements in chapter 4.

Syslog Protocols

Syslog protocols are protocols which aim to secure log data in transit.

The original Syslog (15) is a protocol design which transports log messages from a logging machine to a server, usually called a “collector” or “syslog server”. In its original implementation, several security issues arose which was the reason for the many extensions of the protocol.

Some of its main flaws are the usage of the unreliable UDP network protocol, a missing authentication process between the logging machine and server, the transmission of messages in plain text, and the missing data integrity.

Standard protocols which cope with these flaws can be found in (16) and (17). According to Accorsi in (5), both of these protocols provide origin authentication, integrity and reliable delivery. In addition, the reliable syslog defined in (16) implements confidentiality as well.

Confidential and Searchable Log data:

Within this class of logging approach enhancements the authors of (18) and (19) defined solutions to create a searchable encrypted audit log, allowing to search with specific key words contained in the encrypted log. These enhancements can be combined with all secure logging approaches as they only deal with confidentiality and searchability.

3.3.6 Summary

All approaches described in this document have their advantages and disadvantages. A brief overview is given in this section.

The following two tables summarize the analyzed logging approaches with regards to security. The symbols used in table 3.1 denote: + = covered, - = not covered, (+) = partly covered, N/A = Not Applicable.

Attack	Bellare & Yee	Schneier & Kelsey	Holt	Ma & Tsudik
Insertion	+	+	+	+
Deletion	+	+	+	+
Total Deletion	+	+	+	+
Modifying	+	+	+	+
Verification	-	+	+	(+)
Appending	-	(+)	-	(+)
Truncation	-	(+)	(+)	+
Delayed Detection	N/A	-	N/A	N/A

Table 3.1: Attack Resistance of Approaches Analyzed.

Special Properties	Bellare & Yee	Schneier & Kelsey	Holt	Ma & Tsudik
Trusted Server necessary	yes	yes	yes	yes
Detectable which entry has been tampered with?	yes	yes	yes	-
Authenticity	-	yes	yes	-
Confidentiality	-	yes	-	-

Table 3.2: Special Properties of Presented Schemes.

All approaches are based on the FI-MAC scheme from Bellare and Yee. Therefore, they automatically guarantee forward integrity (see first four attacks).

Notably, all logging approaches have problems with missing mechanisms in order to detect appending attacks. Detection can only be realized if the file has been closed, and therefore no keys are available to build new signatures.

Another major vulnerability is represented by the truncation attack. Each approach tries to deal with this problem in its own way. The only approach presented in this document which successfully detects truncation is provided by Ma and Tsudik. This is achieved by holding the current amount of existing entries in a FssAgg tag. However, the disadvantage is that their scheme does not reveal modified entries. The verification process given can only state whether the log has been tampered with or not. This is a huge drawback if this facet is desired for the logging protocol.

In secure logging approaches based on symmetric cryptography, the attacker gains control over the key. Therefore, these approaches no longer guarantee authenticity.

Holt's Logcrypt provides a public verifiable scheme which separates log creation from verification through the use of asymmetric cryptography.

4 Requirements Analysis

This chapter details the security requirements that must be met in order to enhance the Omega Pro copying process, so that the data may be used as digital evidence. In addition, hardware and process related functional requirements for a future mode of operation are described. Following this, the current mode of operation of the Omega Pro, which does not support digital evidence, forms the basis of analysis of potential attacks and describes the natural boundaries of the secure logging process – specifying what can be expected by programmatic support and what has to be covered by organizational means.

4.1 Requirements

The major objective of this work is to design a secure logging protocol for the Omega Pro, which allows data that is copied to the hard drive to be used as digital evidence. The logging process employed on the copying machine documents the data which is written to a log file residing on the drive. It is required that a verification tool permits an auditor to check if manipulation took place, or if the log file data is consistent with the files which exist on the optical discs. This check must be possible on every PC with the external drive attached to it.

The above implies that the following security requirements must be fulfilled:

Security Requirements

(R1) Data Integrity:

- a. Data on the external disc cannot be modified without detection by the verifier.
- b. Subsets of data which have been modified are detected, evidence should be provided for the subsets that are valid.

(R2) Log Integrity:

- a. The log file cannot be modified without detection by the verifier.
- b. Subsets of log entries which have been modified are detected, evidence should be provided for the subsets that are valid.

(R3) Authenticity:

- a. Valid proof that the data remaining on the hard drive has been copied by the Omega Pro by an identifiable user.

The following requirements must also be met:

Additional Requirements

(R4) Hardware and Software

- a. The developed software must be integrated into the current copying machines. This implies that the software must run on the current Operating Systems Linux and Windows XP.
- b. The Omega Pro is the only trusted machine available. All other machines are considered to be untrusted.

(R5) Resource Efficiency:

- a. The overhead generated by the secure logging processes implemented in the system should be within acceptable boundaries with regard to the time used for copying the optical media. Due to the optical disc reading process being very time consuming, the media should only be read once.

(R6) Copy Modes:

- a. Allowing copying in file copy mode: All files on the optical disc are copied one after the other.
- b. Allowing copying in raw copy mode: The disc is copied as a raw disc image.

4.2 The Current Copying Process

When CDs or DVDs have been confiscated by the authorities, the analysis of all the media becomes cumbersome and time-consuming if it is not first copied to a hard drive. One would literally have to go through each disc one at a time at a speed dictated by the optical disc drive. The benefits of copying this media to a hard drive are that the media is easier to share with multiple people, it's much faster, and as a result more time efficient.

In order to make this copying process more productive, German authorities use the Omega Pro (as shown below).

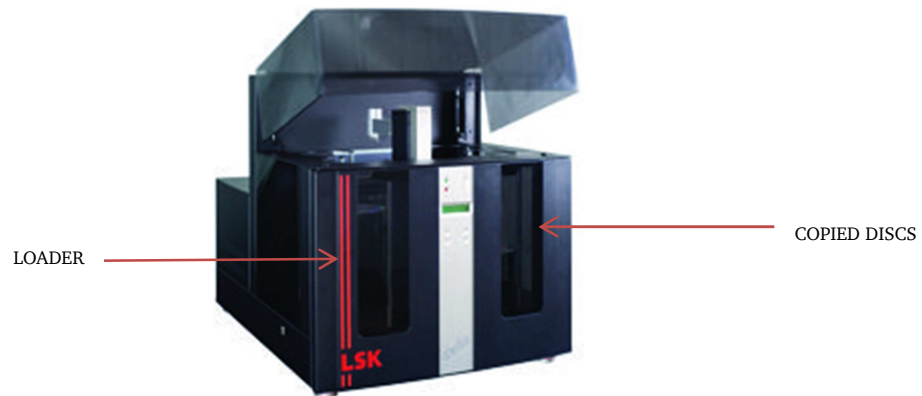


Figure 4.1: Omega Pro

The system is a computer with one or many optical disc-drive(s). The user must put the stack of discs into a loader from which each disc is picked up by a robotic arm and fed into an empty drive. The user then sets the device to start duplicating all the data, either file by file or as raw disc images. The data of each disc is then written onto an external hard drive. This external device is then used for forensic purposes. Once the copying process is complete, the disc is ejected, picked up by the robotic arm, and separately piled.

4.3 Process Phases

The figure below presents the high level phases of the copying process. As shown in Figure 4.2 the process can be divided into three phases which are defined in the following:

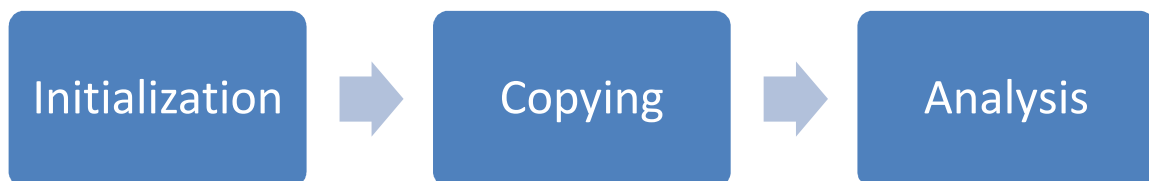


Figure 4.2: Process Phases

Initialization Phase

In the initialization phase, the user sets up the device by putting in the stack of discs and entering information required by the copying process (example: file or raw copy mode, target directories on the external drive).

Copying Phase

The entire process of duplicating disc data, and documenting it via a log file, is called the copying phase. Each disc is read in entirety and duplicated onto the hard drive.

Analysis Phase

As soon as the copy phase is complete, the drive can be removed and evaluated in the analysis phase by multiple users. This phase includes the verification process which allows for determining the data's integrity.

4.4 Attacks and Boundaries

The following provides an analysis and evaluation of possible attacks which could be executed during each of the phases mentioned in the previous chapter. This approach helps determine which attacks the future mode of operation could guard against. It also covers the organizational means which have to be implemented in order to use the copied data as digital evidence.

Initialization Attacks: These kinds of attacks are user-driven and are distinguished by the user's intention. Two scenarios are possible:

- **Removing Discs:** It is the operator's intention to support the accused criminal by removing confiscated discs from the stack.
- **Inserting or Exchanging Discs:** It is the operator's intention to incriminate the accused criminal. Therefore, discs with incriminating content are inserted into the stack. The copying device then writes the information onto the hard drive for further analysis.

None of the scenarios described above can be handled by any system copying a stack of discs. The logging system has no control over a user's behavior which is the reason why it cannot determine if the entered discs are forged or not.

Copying Attacks: Attacks are executed by malware which is located on the copying machine. The malware can manipulate the data while it is running through the memory of the copying device. Such an attack requires knowledge about the system developed in this project. Without the knowledge of this active software, forged data remains undetected. Therefore, this work requires the Omega Pro to be a trusted machine.

Analysis Attacks: A user analyzing the data on the hard drive, can try to tamper with the information by either adding new files to support the accused or change/remove an arbitrary amount of existing files. In order to do so, the attacker must also manipulate evidence in the attached log file, for which, she can apply the log attacks as described in chapter 3.2. Sheltering against these attacks can be done by using verification tools as described in the following chapters. As long as the verification tool is used in a public environment, replacing the verification tool cannot be detected. This threat has to be addressed by organizational means.

The following two analysis attacks need special attention:

Verification Tool Attacks: An attacker manipulates the verification tool so that it accepts forged logs as valid.

Certificate Attacks: When using digital signatures in context to secure logging without a PKI or a similar trusted functionality, an attacker can use her own public key and a log signed with her own private key. A verifier using the corresponding public key will not detect this kind of attack.

Physical Failures: All these scenarios assume that confiscated discs, as well as the hard drive, are free of physical failures.

Hard drive failures happening after the copying process, will result in the file, making use of the damaged block, getting a different hash and marking it as changed. This can be dealt with by using Raid protected hard drives (see in (20)), which logically repair the hard drive.

Optical media failures result in the sectors not being readable anymore. They are not copied to the hard drive.

Organizational/Technical Means: In order to guard against possible attacks, the following must be handled from an organizational point of view (example 4 eyes principle):

- (A1) The operator of the copying unit has to be reliable regarding initialization attacks. It is assumed that the confiscated media is complete and accurate before copying.
- (A2) With regards to the secure logging architecture (as explained in chapter 3.1), the copying unit is considered as the logging machine. It must be made sure that the copying machine is trusted.
- (A3) By introducing organizational means (example 4 eyes principle), the verification system has to be checked with regard to being the original. The concept described in this work guarantees that the verifier can use the correct verification tool. However, the concept cannot guard against the operator using a manipulated tool.

5 Concept

In this chapter, a concept is described addressing the needs of the Omega Pro copying process which supports digital evidence of the copied data. First, a high level design of a future “system” is presented, concentrating on the two main processes, copying and verification. This is then the basis for a description of the required architecture supporting this design. Afterwards, the secure logging principles guarding against all the attacks mentioned in the previous chapter are described, and how the chapter 4 requirements meet the design is explained.

5.1 Processes and Architecture

The following two sections give a high level overview of the two main processes needed to implement secure logging in conjunction with the Omega Pro copying process. The first process deals with copying data from an optical disc to a hard drive, and the second covers the verification of the copied data in relation to the log produced during the copying process.

In his work, Holt presented an approach (see chapter 3.3.4), which made a log file publicly verifiable through the use of public key cryptography. Due to this advantage, the processes described in the following are based on the usage of public and private keys.

The high level functional design which is developed in the following aims at guarding against all the attacks mentioned in chapter 4.4. In the following, the implementation of this design is called “system”. For differentiation purposes, the implementation described in this work is called “prototype”.

To be able to test and evaluate performance impacts of using different ways of signing logs, the prototype allows for evaluating the impact of symmetric cryptography as well.

Due to design decisions made in the beginning of this work, not all required functions (able of guarding against all chapter 4.4 attacks) could be implemented in the prototype. This is further explored at the end of this chapter.

5.1.1 Copying Process

The following diagram provides a high level overview of the proposed copying process:

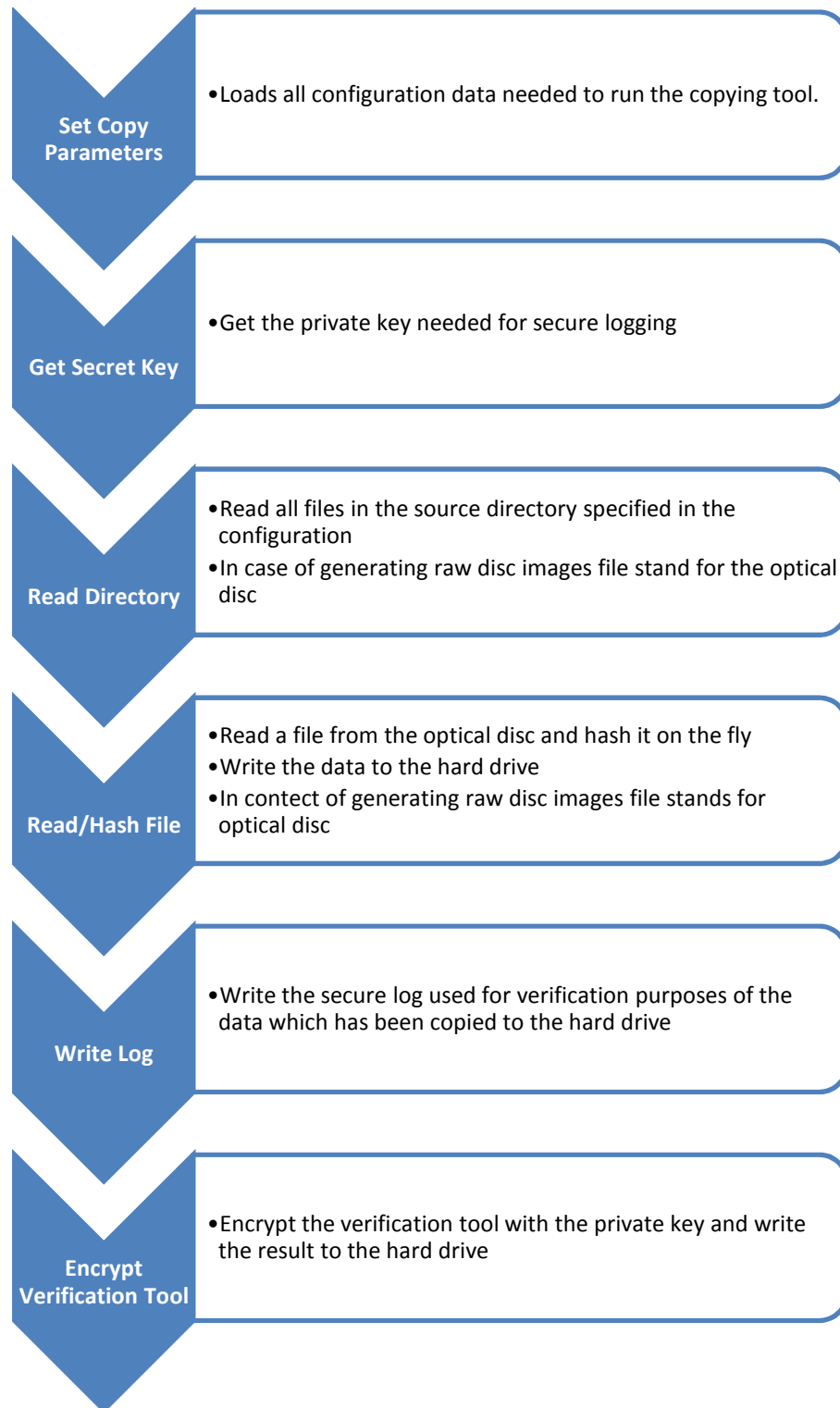


Figure 5.1: New Omega Pro Copying Process

5.1.2 Verification Process

The view of the high level verification process can be seen in the following diagram:

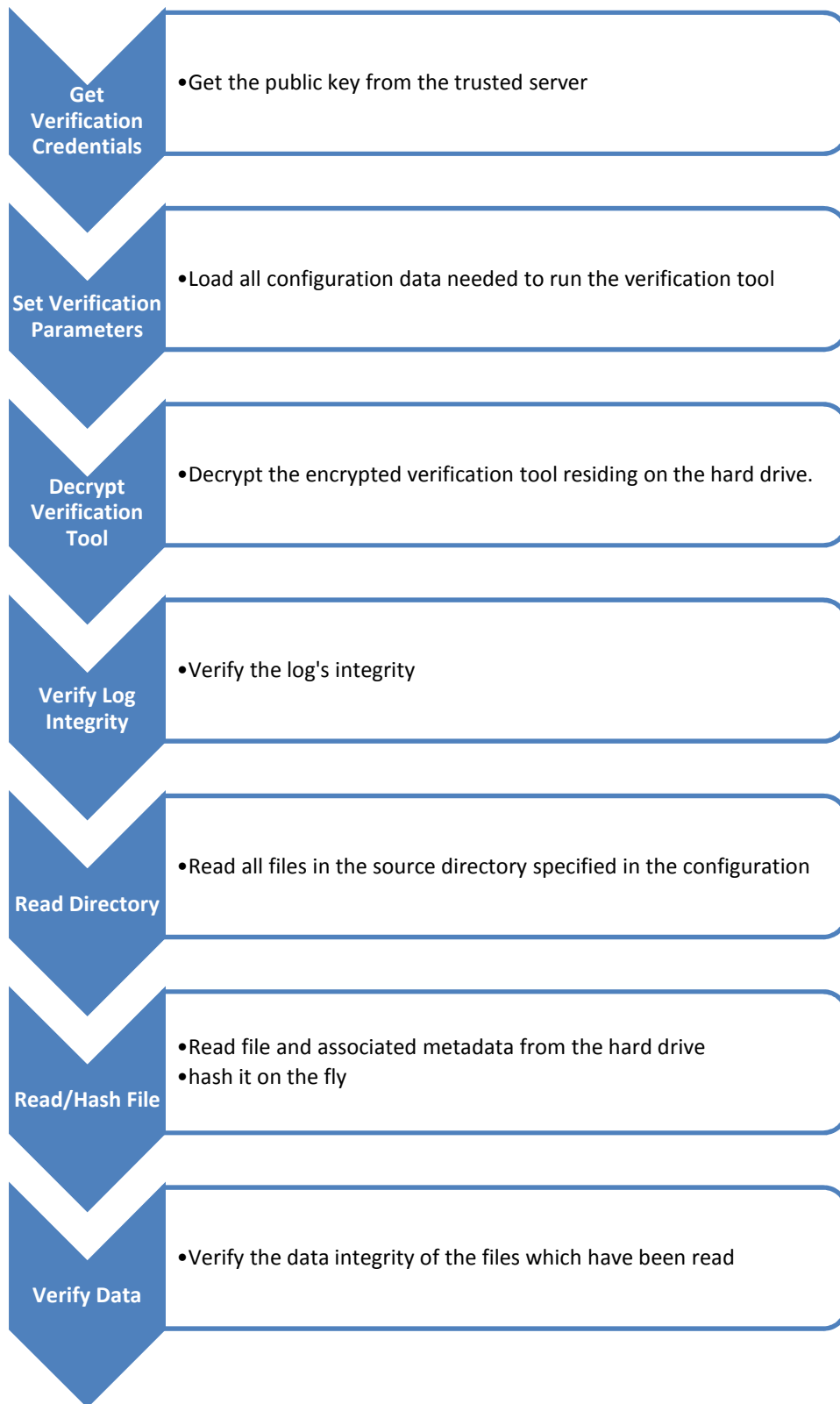


Figure 5.2: New Omega Pro Verification Process

5.1.3 System/Prototype Architecture

The system architecture needed to support the processes in the previous chapter is shown in the following diagram:

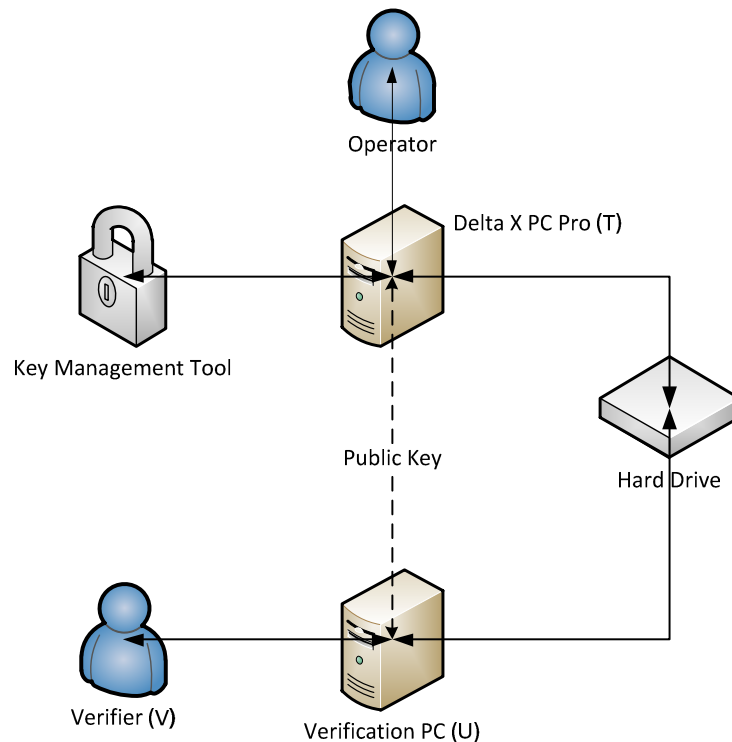


Figure 5.3: System Architecture of the New Concept

The trusted Omega Pro T steps into the role of the logging machine. The same server hosts a key management tool allowing the maintenance of public and private keys. In the prototype, a key database supports the maintenance of these keys in a password protected environment. In order to support authentication (R3), the system makes use of private keys stored in the key database, which allows identifying the users on an alias (name)/ password level.

The verifier V accesses the untrusted verification PC U , which is linked to the same hard drive as T . T supplies the public key needed to allow V to verify the data on the hard drive.

The prototype architecture corresponds to the above with the exception that the dotted line between T and U does not exist. The prototype was developed on one machine only. To simulate the functionality shown above, a public key certificate is generated using the key management tool.

It should be noted that due to this design, the functionality provided by the prototype does not protect against verification tool attacks and certificate attacks as described in chapter 4.4.

5.2 Secure Logging Principles

A log file is generated according to the following principles to cover the system requirements described in chapter 4:

1. Files logged are assigned a sequential number which allows the detection of missing log entries (R2).
2. A hash is created and logged for every file read from the optical media which:
 - a. Allows the identification of manipulations to the file and
 - b. Is the means to achieve data integrity for the information stored on the hard drive (R1).
3. Every log entry contains the date last modified of the file. This allows detection of changes made to this metadata.
4. Every log entry is signed which allows the detection of changes made to the log entries (R2).
5. The log, consisting of all log entries and signatures, is signed to guard against deletion, truncation or insertion attacks (R2).

5.3 Signing Methods

From a requirement's perspective, using digital signatures is the best option because the disadvantage of a symmetric scheme is that the private key (linked to one person) is used in an untrusted environment. This problem is avoided by using public key cryptography.

Analog to the state of the art schemes, the log's integrity is assured by marking each entry with either a digital signature or a MAC (prototype only). This follows the ideas presented in the Bellare and Yee symmetric and Holt's Logcrypt asymmetric approaches.

Due to the fact that the generated log is finite (limited by the number of files on one disc), forward integrity aspects do not play an important role for the design of a future system. Nevertheless, this alternative was implemented in the prototype for analysis purposes.

5.4 Summary

The following summarizes how the security requirements are met by the concept described in the previous chapters.

Security Requirements

By signing the log entry and the log (as described above) it is possible to cover:

-
- (R1) Data integrity is assured by hashing the file and making the hash part of the log entry.
 - (R2) Log Integrity is assured by signing every log entry and the log as a total.
 - (R3) Authenticity is assured by user authentication when accessing the key database, and by the (public key) user authentication when verifying the log.

Other Requirements

Details on how to fulfill other requirements will be covered in the following chapter implementation. Listed below is a summary.

- (R4) Hardware and Software:
 - a. Choosing Java as the basis for all development work guarantees that the developed software runs on the operating systems Linux and Windows XP.
 - b. The Omega Pro is used as the only trusted server.
- (R5) Resource Efficiency:
 - a. This is guaranteed by using hash on the fly technologies. The test results shown in chapter 7 prove that the overhead for secure logging are within acceptable boundaries.
- (R6) Copy Modes:
 - a. Implemented in the prototype.
 - b. Implemented in the prototype.

Furthermore the process implemented guards against the following attacks:

Modifying, insertion, reordering, deletion, truncation, total deletion and appending attacks are detected during the verification by checking the log signature. The logging principles described in chapter 5.2 identify which information has been forged.

Delayed detection attacks are not applicable in context to the copying process of the Omega Pro. The log is generated in one go, then closed, and written to the hard disc.

Verification attacks require the use of symmetric cryptography. This concept foresees the use of asymmetric cryptography. Therefore, verification attacks are not possible.

Certificate Attacks cannot be carried out as the verifier receives the public key from the trusted Omega Pro. It is clear, that the transmission between trusted and untrusted server must be secured.

The concept described in this chapter guards against verification tool attacks in the following way: after the copying process, carried out in the trusted environment, the verification tool is copied to the hard drive in an encrypted form. When starting the verification, it is decrypted using the public key supplied by the trusted server. This makes it impossible for an attacker to manipulate the verification tool without knowing the private key.

Summarizing the above:

- The prototype developed in this work meets almost all requirements. It is vulnerable to certification and verification tool attacks.
- The concept for the system described in this document fulfills all requirements.

Therefore,

- The functionality of the prototype's copying and verification can be used for digital evidence.
- By using the prototype the productivity of the investigating authorities is increased.

6 Implementation

This chapter gives an overview of how the prototype titled “CopyDVD” has been developed and implemented. This includes:

- The development environment including the libraries used.
- An overview of the main components of the prototype.
- The implementation principles applied to meet the required functionality most effectively.
- An overview of the package structures and the classes within the packages.
- An overview of all cryptographic methods used to develop the required functionality.
- A description of the log structure used in this prototype.
- A description of the logic used to implement the verification processes.
- An overview of the graphical user interface and a description of the prototype setup.

6.1 Development Environment

Due to the requirement (R4a) - usability on Windows and Linux operating systems - it was decided to implement the prototype using **Java**².

The CopyDVD prototype was developed using the free software development kit **Eclipse**³ and the **Visual Editor**⁴ plugin on the basis of Java. All plugins and libraries are open source based and freely available.

Tests have been conducted using the **JUnit**⁵ framework.

The class diagrams shown in this section have been generated using the Eclipse plugin **green**⁶. Green is an UML class diagram editor, supporting both software engineering and reverse engineering of existing code.

The prototype makes use of the following Java libraries:

- General events, timer information, debugging information or warnings are logged and outputted with the Apache Software Foundation framework **log4j**⁷, a framework that has been developed without creating high performance costs.

² Version 1.6: <http://java.sun.com/>

³ Version 3.6.0: <http://www.eclipse.org/>

⁴ Version 1.5.0.R20101202-1328: <http://www.eclipse.org/vep/>

⁵ Version 4: <http://www.junit.org/>

⁶ Version 3.5.0: <http://marketplace.eclipse.org/content/green-uml>

⁷ Version 1.2.16: <http://logging.apache.org/>

log4j allows log messages to be written in customized formats. These messages can be written onto the console, as well as to files, using different appenders.

- Configuration information for the prototype is stored in the Windows .ini format. The prototype makes use of the open source library **ini4j**⁸, a Java API for handling configuration files in Windows .ini format. This library provides handy methods to quickly read out these files and embed the content into the program.

6.2 System Modules

The main modules of the CopyDVD system are shown in figure 6.1. In context of the figure “module” is denoting a class or set of classes that perform main functions. “Components” in this context are functions within a module.

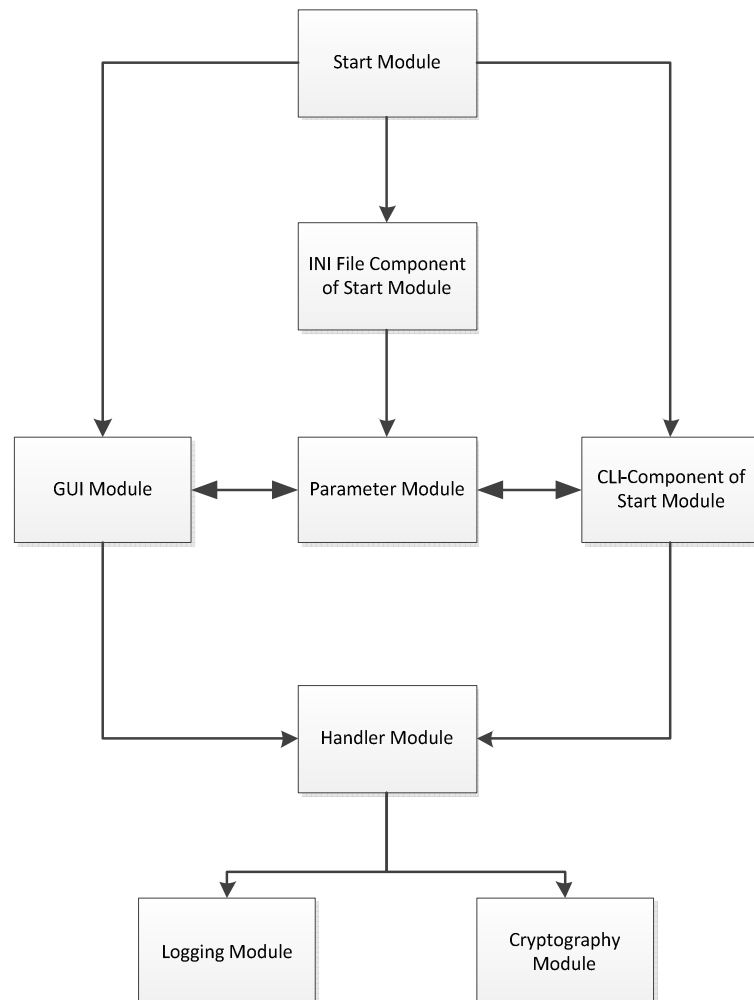


Figure 6.1 System Components

⁸ Version 0.5.2: <http://ini4j.sourceforge.net/index.html>

The **start module** is the main module for running the system. Via the .ini file component of the start module, it loads the parameters that are stored in an external .ini configuration file into the **parameter module**, the repository of all system parameters. Depending on the number of CLI parameters passed to the start module, the GUI module is called or the CLI component of the start module continues to process the commands passed over.

The **GUI module** is the graphical interface to the user and allows for modifying the cryptographic and system parameters that have been set when loading the .ini file.

A major portion of the program logic is controlled by the **handler module**. Depending on the verifying or copying tool described in the following chapter this module handles the communication with the key database, all calls to the cryptographic module, and the logging module.

The **cryptographic module** is implemented using a factory design pattern delivering the MAC and digital signatures needed to sign the logs.

The **logging module** takes care of writing the logs (in the case of copying), and reading the logs in the case of verifying.

6.3 Implementation Principles

Following the design described in chapter 5, the following tools are needed:

A copying tool that operates in the trusted environment on the Omega Pro, and a verification tool that can be used in an untrusted environment. The main functions of these tools are:

1. **Copying**- optical media content to the external hard drive
2. **Verifying the Log Integrity** - checking whether the log has been manipulated
3. **Verifying the Data Integrity** - checking whether the data on the external drive has been manipulated and precisely matches the content of the log supplied

The copying tool is implemented in a way that all functions can be executed. The verification tool allows executing the functions 2 and 3 only.

In order to minimize programming effort, the following principle was applied: the verification functions are implemented completely in an abstract class. In this class, all methods required for copying are abstract. The verification classes extend the abstract classes with empty bodies. The copying tool classes extend the abstract classes with copying logic added. An example of how this has been implemented is shown in the handler module in the following figure:

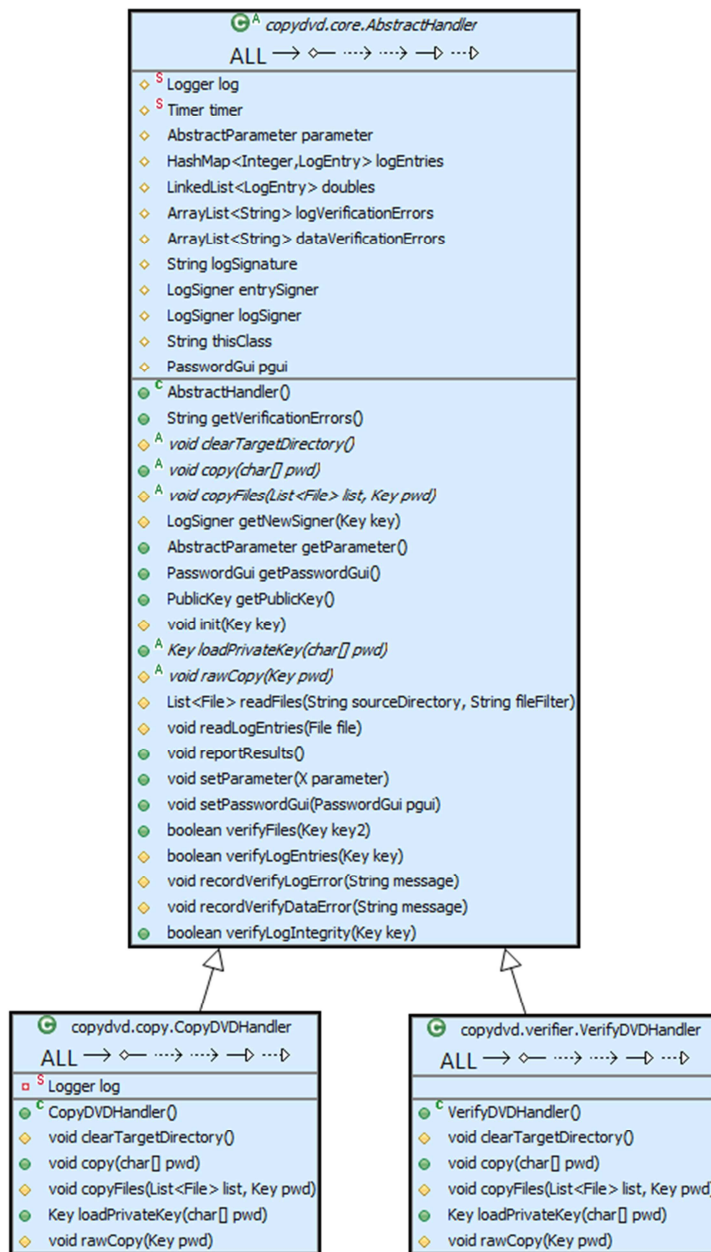


Figure 6.2 Sample Copy/Verify Design Principle

In order to run tests with different signing methods using the same code, a factory design was implemented. Using this design easily allows switching between signing with digital signatures or MACs (with or without forward integrity). The following figure shows this in form of a class diagram.



Figure 6.3: Log Signer Factory

6.4 Packages

The package structure chosen supports building the copy and verification tools. It is shown in the figure below.

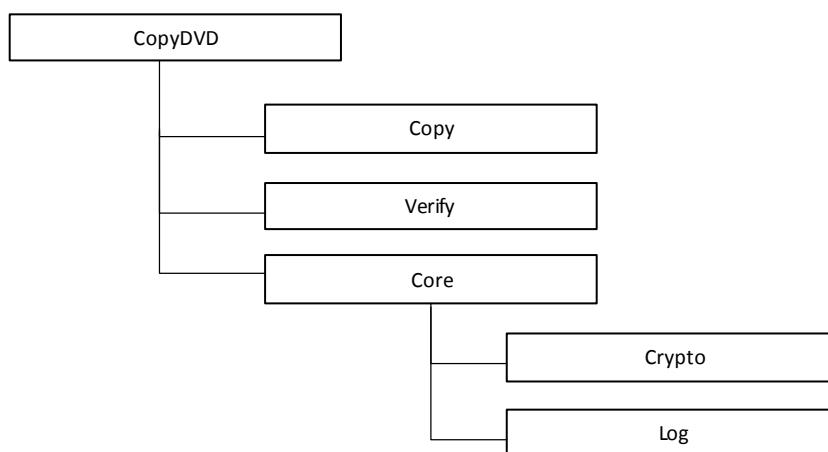


Figure 6.4 Package Structure

The core package, including its sub packages, hosts all classes required by both tools. By adding the copy or verify package to the core packages, the copying or verification tools can be built.

6.5 Class Descriptions

The following describes the classes developed. Note that all classes starting with “abstract” already implement verification logic, and leave out any copying logic.

Package `copyDVD.core`:

- **AbstractGui.java:** This abstract class provides the main functionality of the GUI module. It is responsible for interacting with the user and communicating with the parameter and handler module.
- **AbstractHandler.java:** This class implements methods to verify the integrity and authenticity of a log and copied files.
- **AbstractParameter.java:** A data structure used to maintain all tool specific parameters.
- **AbstractStart.java:** AbstractStart initializes the program at start.
- **AliasDialog.java:** Dialog class used to enter the user’s alias.
- **FileFinder.java:** A file crawler to filter all files which match a regular expression in a given directory. In addition, it can be used to delete an entire directory.
- **GenericFileFilter.java:** A file filter class used in conjunction with JFileChooser to select, for example, .log or .ini files.
- **CopyDVDException:** This exception is used to detect errors when entering incorrect information in GUI text fields allowing reentering the information without aborting the program. All other exceptions thrown by the GUI will abort the program.
- **PasswordGui.java:** Provides a GUI enabling a user to enter his/her password.
- **Timer.java:** This class provides methods for chronometry purposes.
- **LogEntryTimer.java:** This class is used in conjunction with measuring the duration of program function reading, writing, copying, and cryptographic calculations.

Package `copyDVD.core.crypto`

- **HashReader.java:** This class implements methods to read data from an optical disc, write it to the target directory, and simultaneously build a hash on-the-fly.

-
- **LogSignerFactory.java:** Based on the factory pattern, this class delivers objects of either the LogSignerMac or LogSignerSignature class.
 - **LogSigner.java:** An interface class which provides methods to sign and verify a log's content. This interface is implemented by the LogSinerMac and LogSignerSignature classes.
 - **LogSignerMac.java:** This class provides all functionality to sign and verify messages with MACs. In addition, signing keys can be evolved for forward integrity purposes.
 - **LogSignerSignature.java:** This class provides all functionality to sign and verify messages with digital signatures.

Package copyDVD.core.log

- **LogEntry.java:** The data structure defining a log entry.
- **LogEntryReader.java:** Reads all log entries out of a log file.
- **LogEntryWriter.java:** Writes log entries onto a log file.

Package copyDVD.copy

- **StartCopyDVD.java:** Derived from the AbstractStart.java class. It hosts the main method to start the copy tool. Depending on the number of parameters passed to the main method, it processes CLI-commands or initializes the CopyDVDGui.
- **CopyDVDGui.java:** Extends its abstract class by enabling the copying feature for a user.
- **CopyDVDHandler.java:** This class extends the AbstractHandler and implements the copying methods needed to copy optical media.
- **CopyDVDParamter.java:** The AbstractParamater data structure is extended with additional functionality to maintain key database parameters.

Package copyDVD.verify

- **StartVerifyDVD.java:** Derived from the AbstractStart.java class, it either processes CLI-commands for verification, or initializes the VerifyDVDGui.
- **VerfiyDVDGui.java:** All functionality is already implemented in its abstract class. For this reason, the bodies of abstract methods remain empty.
- **VerifyDVDHandler.java:** All functionality is already implemented in its abstract class. For this reason, the bodies of abstract methods remain empty.

- **VerifyDVDParamter.java**: All functionality is already implemented in its abstract class. For this reason, the bodies of abstract methods remain empty.

6.6 JUnit Test Cases

The prototype was developed using JUnit Test cases allowing testing of the essential functions of the prototype in a very efficient way, helping to maintain the quality of the code after having made changes to the prototype.

6.7 Cryptographic Algorithms Used

The prototype tools have been developed using standard Java cryptographic packages. Cryptographic methods⁹ used for the prototype can be seen in the following table:

	Symmetric	Asymmetric
File Hashes	SHA-1	SHA-1
Log Entries	HMAC SHA1	SHA1withRSA
Logs Files	HMAC SHA1	SHA1withRSA

Table 6.1: Cryptographic Algorithms Used

Initially RSAwithMD5 was used to sign the log entries and the logs. The results of the tests shown in chapter 7 supported the switch to SHA1withRSA.

6.8 Log Structure

Let L_i be the log entry of the i .th file F_i , $L' := L'_1 L'_2 \dots L'_n$ the log as defined in chapter 3.3.1, and Z the signature of L' . If data is copied in raw copy mode, the raw image is treated as one file.

The structure of L_i is shown in Figure 6.5.



Figure 6.5: Log Entry Structure

⁹ <http://download.oracle.com/javase/1.5.0/docs/guide/security/CryptoSpec.html>

The main component of a log entry is the hash value of F_i . In order to associate and indisputably identify a file via its corresponding log entry, additional information needs to be specified:

- i denotes a sequential number assigned to detect missing log entries.
- T_i denotes a timestamp which marks the time F_i has been copied.
- SP_i denotes the absolute source path of F_i .
- DP_i denotes the relative destination path of F_i .
- sz_i the size of the copied file.
- DLM_i is the data last modified which exists on the optical disc.
- $D_i = H(F_i)$. This is the hash built during copying.

In order to illustrate the structure of a log created by the prototype, an example of a raw copy log is shown below:

```
<Entry>
  <Counter>1<\Counter>
  <Timestamp>23.7.2011 || 16:22:10 (+889 ms)<\Timestamp>
  <FileIn>\\.\Y:<\FileIn>
  <FileOut>c:\CopyDVD\CopyTarget\Copy\rawCopy.iso<\FileOut>
  <Size>0<\Size>
  <DateLastModified>19416857022935<\DateLastModified>
  <Hash>0f82b50a456270b0e5b6e45f4275aaab5997e03d<\Hash>
<\Entry>
<EntryAuthentication>bb60d7abfc7ff5d83b367a4b32ee534e81554de9<\EntryAuthentication>
<LogAuthentication>3e500f38aac87c4472974198d822245b0f48edaa<\LogAuthentication>
```

As can be seen, the log entries are built using xml tags which enable direct and simple reading access of a log file. Each log entry has an attached signature. Once the log is written, it is closed by the log signature Z .

It should be noted that the prototype checks the content of the log as described in the following:

- Entries are accepted as valid if the content of a log entry is valid from a syntactical point of view. Content between ending tags and beginning tags is ignored.
- Every `<Entry>` `<\Entry>` must be followed by an `<EntryAuthentication>`.
- `<Entry>` and `<EntryAuthentication>` blocks can be toggled within the log as long as they are toggled together.
- The last `<LogAuthentication>` `<\LogAuthentication>` read is used to verify the content in the `<Entry>` and `<EntryAuthentication>` blocks.
-

6.9 Verification

The data residing on a hard drive is analyzed using the verification tool. Data and log verification needs to be distinguished.

Verifying Log Integrity

Depending on the signing method which has been used, verification distinguishes by either using the private MAC key or a public key. The following pseudo code describes the verification of a log file for the asymmetric case.

Given

Public Key pub to verify the signatures.
List $L' = L'_1 L'_2 \dots L'_n$ a list of log entries with entry signatures
LogSignature Z, the signature of log L'

Algorithm

```
VerifyLogIntegrity() {
    Boolean logIntegrity = true;
    int seqNr = 0;
    for each  $L'_i$  in  $L'$  do
        increment seqNr;
        while (i does not match seqNr) do
            "Log entry  $L_{seqNr}$  is missing"
            logIntegrity = false;
            increment seqNr;
        endwhile
        if (verifySign( $L_i, z_i, pub$ ) != true) do
            "Log entry  $L_i$  has been modified or inserted"
            logIntegrity = false;
            remove ( $L_i, z_i$ ) from list  $L'$ ;
        endif
    endfor
    if (verifySign( $L', Z, pub$ ) != true) do
        "Log has been modified"
        logIntegrity = false;
    endif
    return ( $L', logIntegrity$ );
}
```

Each log entry, as well as the signature of the log is checked for its authenticity. This method returns a set of valid log entries and a Boolean value which indicates whether verification was successful or failed. Verification fails if, and only if, at least one of these signatures do not match. In this case, these entries disqualify for data verification and are therefore removed from the list.

Verifying Data Integrity

Data is verified depending on the list and the Boolean value which are returned by the log verification function. Data verification can only be successful if, and only if, the VerifyLogIntegrity-method returns ($L', true$).

The following describes the logic applied.

Given

List M = list of files F containing all files from a target directory

Algorithm

```
VerifyDataIntegrity( ) {
    Boolean dataIntegrity = true;
    Boolean logIntegrity;
    LogFile L';
    (L', logIntegrity) = VerifyLogIntegrity( );

    for each Li' in L' do
        if (M contains file F with path DPi) do
            if (Di' != Hash(F)) do
                "File F has been modified"
                dataIntegrity = false;
            endif
            remove F from M;
        else
            "File F does not exist"
            dataIntegrity = false;
        endif
    endfor
    if (M is not empty and logIntegrity == true) do
        "All remaining files in M have been added."
        dataIntegrity = false;
    endif
    return dataIntegrity;
}
```

The first if-statement determines whether the file exists on the hard drive or has been deleted. If it was found, the second if-statement determines whether the file has been modified or not. If all log file entries have been processed and the remaining map is not empty, files have been added spuriously.

6.10 Graphical User Interface

In this section the main functions of the GUI of the copying and verification tools are explained. Both tools are started using the configuration .ini file as a parameter. The verification tool is set up using the same abstract GUI, disabling and hiding functions not needed for the verification. Therefore, the verification tool GUI is not described in this document.

6.10.1 Alias Dialog

Once started, the **alias dialog** shown in the figure below asks for the user's alias.

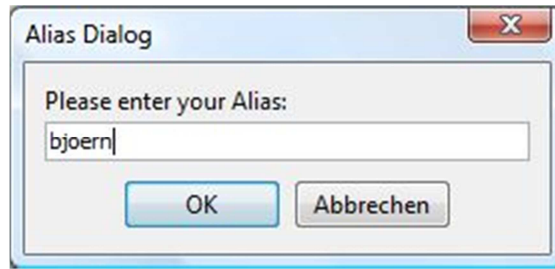


Figure 6.6: Alias Dialog

Provided the alias is in the key database, the main screen pops up (see Copy Tab).

6.10.2 File Menu

The **file** menu

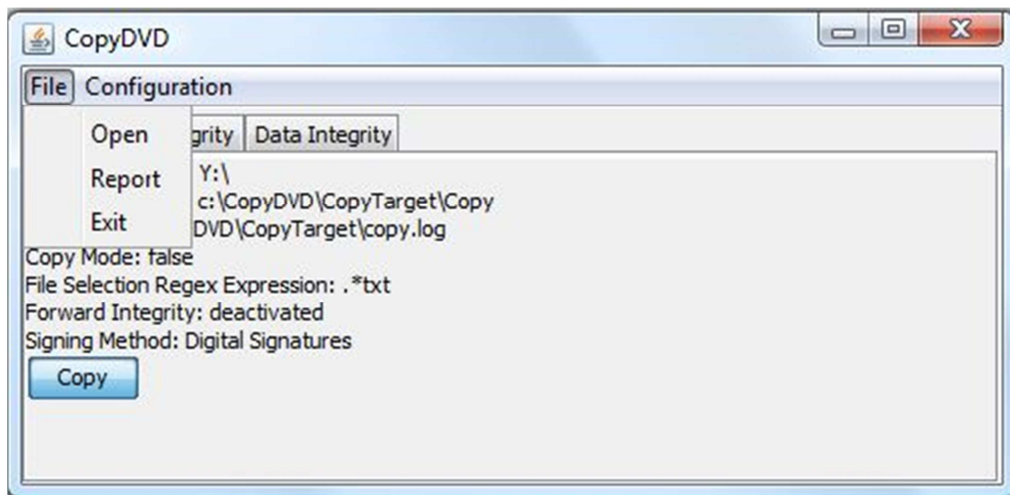


Figure 6.7: File Menu

provides the functionality to **exit** the program, **report** test results of a copying or verification function (note: results are written to the log), and to **open** a new configuration file as shown below.

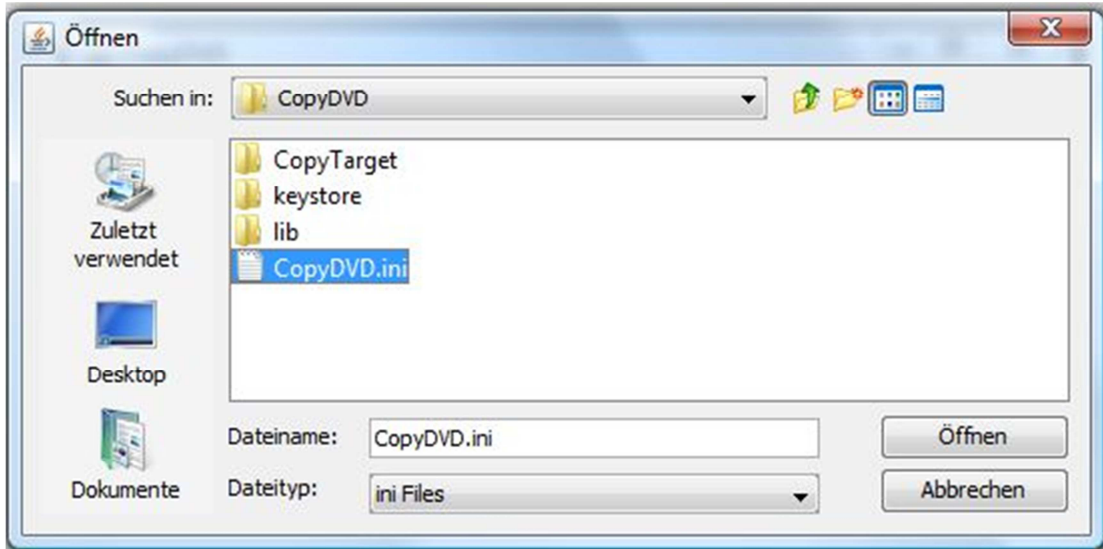


Figure 6.8: Open Configuration File Window

6.10.3 Configuration Menu

The **configuration** menu

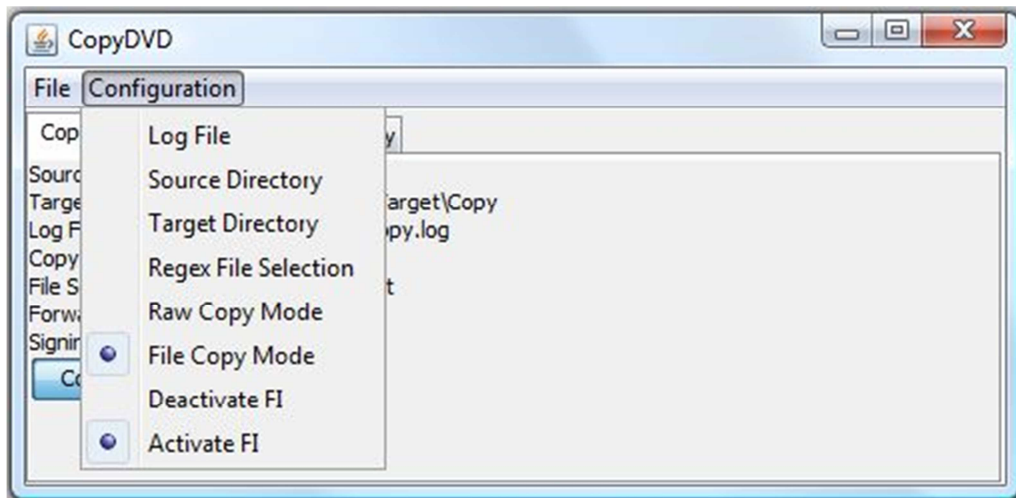


Figure 6.9: Configuration Menu

is used for setting a new **log file** via the **LogFile dialog**,

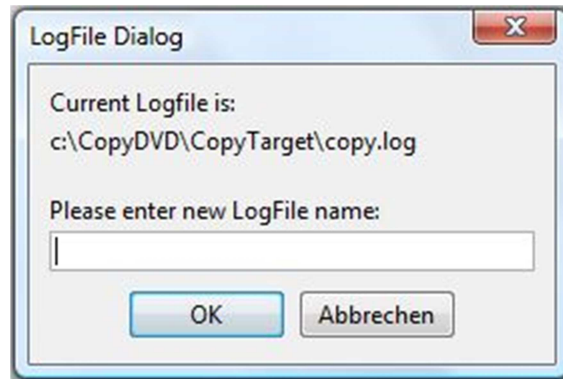


Figure 6.10: Log File Dialog

setting the **source and target directories** (same selection dialog as for the .ini files) of the copying process, and **selecting** which files should be copied. The menu also hosts two radio button groups, one for selecting the **file/raw copy** mode, the other for the **de-/activation of forward integrity**.

6.10.4 Copy Tab

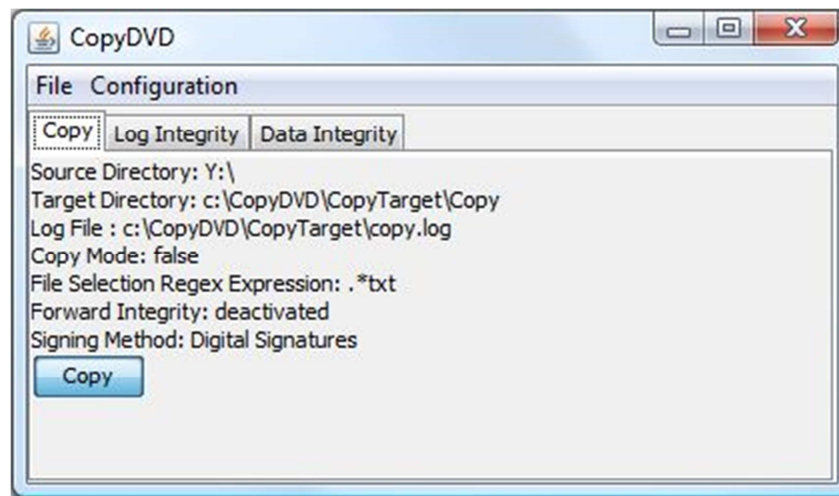


Figure 6.11: Copy Tab

The **copy tab** displays all the parameters that have been set by loading the .ini file. The copy button starts the copying process from the source to the target directory.

After pressing the copy button, a **password dialog** (not shown) allows the users to enter the password needed to read the user's private key from the key database.

6.10.5 Log Integrity Tab

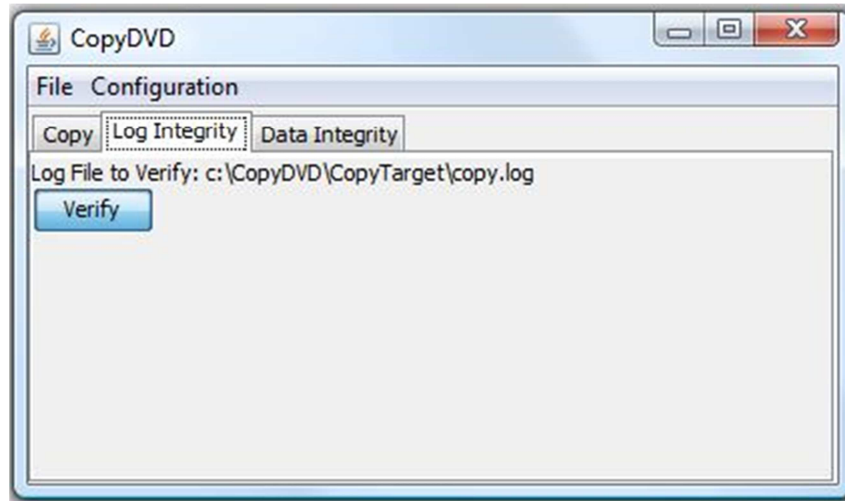


Figure 6.12: Log Integrity Tab

By pressing the verify button, the tab allows the verification of the log file that has been specified. In the case of digital signatures, the verification starts immediately. In the case of signing with MACs, the password must be entered using the **password dialog**.

6.10.6 Data Integrity Tab

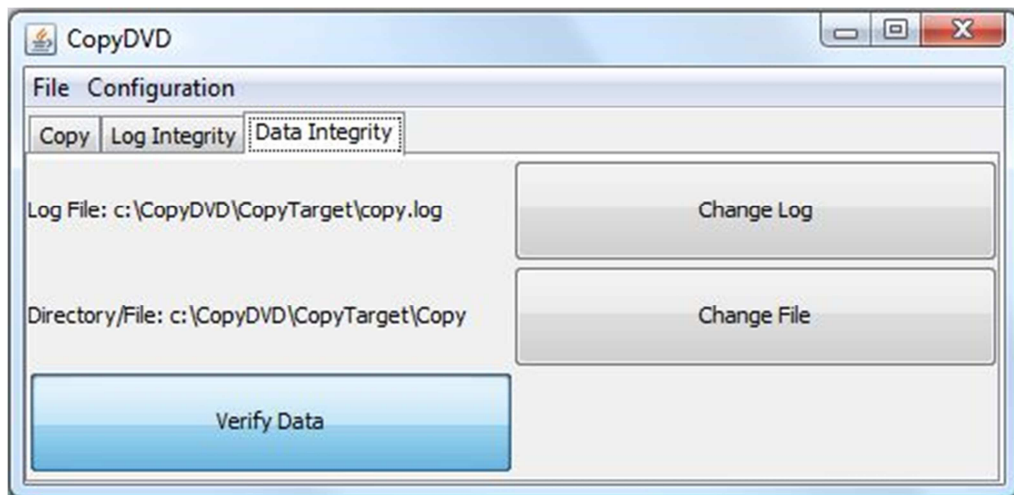


Figure 6.13: Data Integrity Tab

This tab allows the verification of the integrity of the log file versus the target directory shown. All buttons are self explanatory. The password login described in the previous chapter applies as well. Note, this includes log integrity.

6.11 Prototype Setup

The following describes all the components that are used in conjunction with the setup of the prototype.

6.11.1 Key Database

In order to implement the key database, the key and certificate management utility **keytool**¹⁰ supplied by Oracle is used. Personal public/private key pairs and associated certificates can be administered with this utility.

The keytool stores the keys and certificates in a file called keystore. Keystore is protected with a password. In addition, every private key requires entering a separate user (identified by an alias) specific password.

To generate the keystore, the private/public key, and the certificate, two batch jobs are used:

keystore.bat deletes the old keystore and generates a new keystore including a new private and public key for a given alias.

certificate.bat generates the certificate for a given alias.

CopyDVD.keystore is the keystore used for developing the prototype.

Bjoern.cert is the certificate used for the test user with the alias bjoern.

6.11.2 Ini and Property Files

Parameters maintained in the **CopyDVD.ini** file are:

- **Source:** This field denotes the path of the optical disc drive to be read.
- **Target:** This field contains the path to write data to.
- **SecureLog:** All logging messages created by the log4j framework are written onto the log file denoted in the given path.
- **FileFilter:** A regular expression to define the filter pattern in file copy modus. Examples of a filter are “.” in order to copy all files or “*.pdf” in order to copy all files ending with pdf.
- **Forward Integrity:** This parameter is set to “activated” if forward integrity is desired.
- **CopyMode:** Via this parameter, it is distinguished whether data is copied file-by-file or as iso images.

¹⁰ Version 1.4.2: <http://download.oracle.com/javase/1.4.2/docs/tooldocs/windows/keytool.html>

-
- **Store:** The path of the keystore file.
 - **Password:** The keystore password. Note that this is not the alias password.
 - **Certificate:** The location of a public key certificate.
 - **Type:** Sets the encryption method (MAC or digital signatures) for the program.

The **log4j.properties** file contains information steering the logging format, the logging destination and the logging details. The logging level can be set on a single class.

6.11.3 Prototype Tools

The **CopyDVD.jar** file provided covers all the functions of the prototype whereas **VerifyDVD.jar** can be used for verifying purposes only. It should be noted that the verification tool is working on the basis of digital signatures only.

Both tools make use of the log4j and ini4j libraries described in chapter 6.1.

7 Performance Measurements

In order to evaluate the performance impact of the signing methods digital signature and MAC, the design of the system had to cope with both methods. Log entries and logs were always signed with the same method.

7.1 Performance Test Scenarios

To analyze the cost impact of different secure logging approaches the following methods were analyzed:

- **MAC method:** Each log entry and the entire file is signed with a MAC without forward integrity.
- **MAC-FI method:** Each log entry and the entire file is signed with a MAC with forward integrity.
- **Digital Signature method:** Each log entry and the entire file is signed with a digital signature.

In addition, all tests were conducted by copying in two different modes:

- **Raw Copy Mode:** The disc is copied raw disc image.
- **File Copy Mode:** Each file on the disc is copied. The directory structure is retained.

All tests were executed on two different computers with the following characteristics:

Name/Brand	Medion	Dell
Operating System	Windows Vista 32bit Service Pack 2	Windows XP 32 bit Service Pack 3
CPU	Intel Core 2 2.13 GHz	Intel Pentium M 1.86 GHz
RAM	2 GB	2GB
Optical Disc Drive	HL-DT ST DVD-ROM GDR8164B ATA-Device	Philips CDRW/DVD SCB5265

Figure 7.1: Test PC Descriptions

The Omega Pro copies up to 600 discs in one go until it has to be recharged with new sets of discs. When processing large piles of optical media, several optical disc drives run parallel to each other.

Due to the sequential processing of reading optical media in one drive, it can be expected that results scale linear in relation to the number of media per optical disc drive.

Effects of parallel processing using several optical disc drives were not analyzed. As long as these processes use the same CPU, it is expected that this will have an impact on the overall performance.

7.2 Results

The following presents the major findings of the tests conducted. All test were executed using the Junit 4 Java package and were run automatically. Test results were logged and analyzed. The log files used, and the analysis of the results can be found in the accompanying documentation.

Initially, all tests were carried out using the RSAwithMD5 algorithm for digital signatures. Subsequent tests using the SHAwithRSA algorithms revealed that the usage of these algorithms does not impact the overall performance. Therefore, SHAwithRSA was chosen for the prototype.

In order to evaluate the impact of measuring, the timer used in the prototype was switched off and results were compared to each other. The results which have been achieved clearly indicate that the usage of the timer has a negligible impact.

All tests shown in this section have been conducted using a DVD with 4.37GB of content.

To reduce the impact of internal system processes in the evaluation, all the tests were carried out three times and the average was used in the analysis.

7.2.1 Copying

Figures 7.2 and 7.3 show the results of the tests. All results are illustrated in seconds. The legend of the diagram below denotes:

Log Signature: time needed to sign the log file itself.

Log Entry Signature: time needed to sign the the sum of all log entries.

Hash on the Fly: time needed to calculate the hash on the fly for copied data.

Write to Hard Drive: time needed to write the data to the hard drive.

Read Optical Disc: time needed to read the optical media.

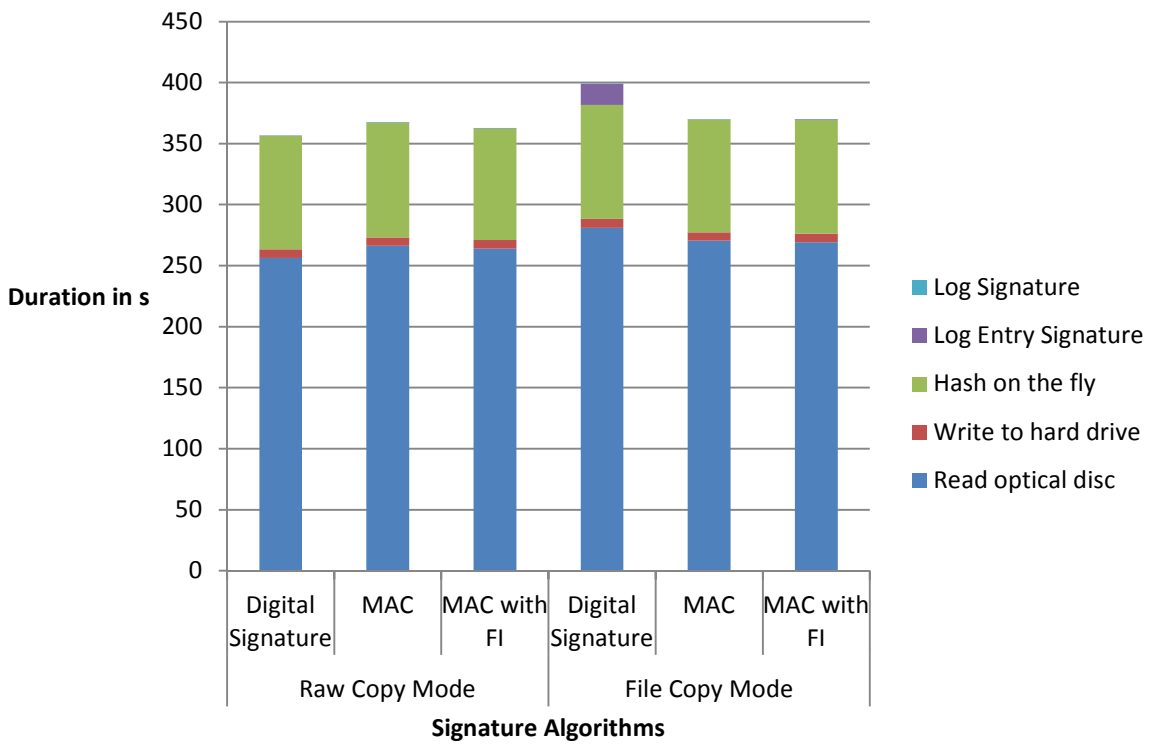


Figure 7.2: Medion Copy Test Results

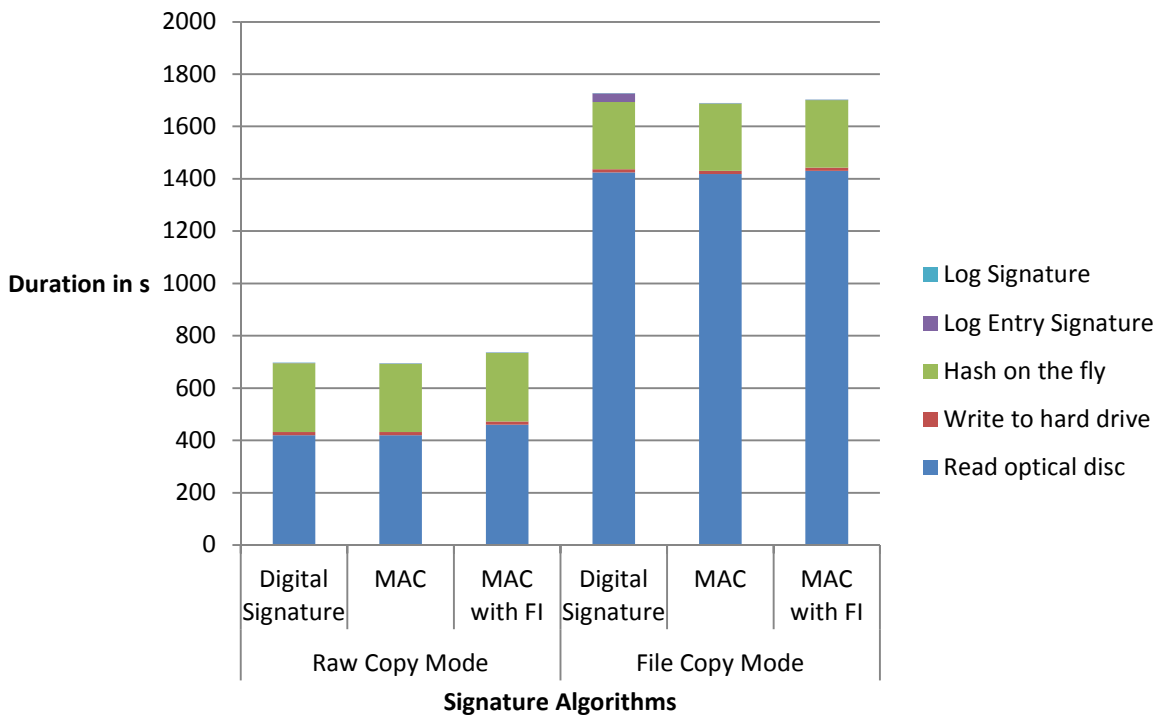


Figure 7.3: Dell Copy Test Results

Results can be summarized as follows:

- In comparison to the older Dell, all Medion results are significantly better, showing the impact of CPU power and optical disc performance.
- On the Dell, the “Read” of the file copy mode differs in a significant way (3 times more) from the results in the raw copy mode. This cannot be seen on the Medion where this difference is negligible. This behavior is due to reaching the limits of the Dell CPU while being in file copy mode.
- Compared to the time used for the other items, the signing of the log entries and of the log add almost no overhead to either PC. The only exception to this is by using digital signatures in conjunction with the file copy mode. The DVD used contains 1700 files, resulting in the same number of log entries that have to be signed. Even in this case, the overhead is minimal compared to the other items.
- Current operation processes (sum of Read and Write) consume most of the time used for copying (see Figure 7.4).

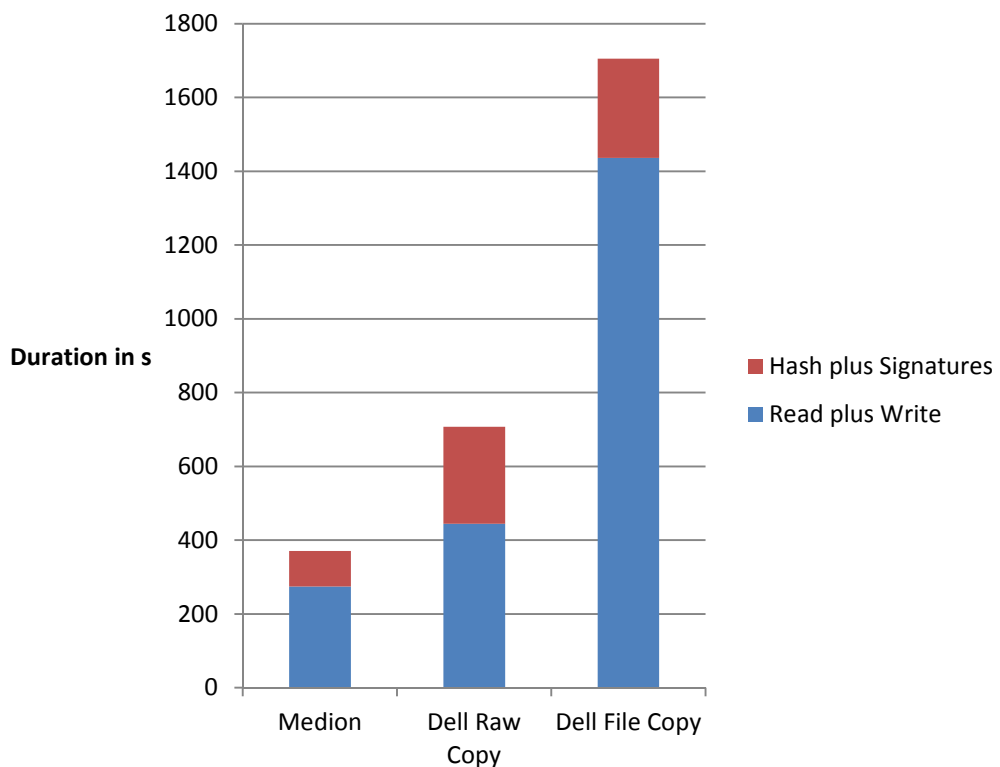


Figure 7.4: Relation of Old and New Functionality

- The overhead of the hash on the fly has the biggest impact of all new cryptographic functions implemented by the secure logging methods. Assuming Read and Write to be the basis (100%), the overhead of the hash on the fly is in the range of 28% to 34%. These numbers should serve

as a direction only, as these percentages are heavily dependent on the speed of the optical drive (determining the basis of 100%) and the CPU used (determining the overhead on the basis of the 100%).

7.2.2 Verifying

The results of the verify tests can be seen Figures 7.5 and 7.6.

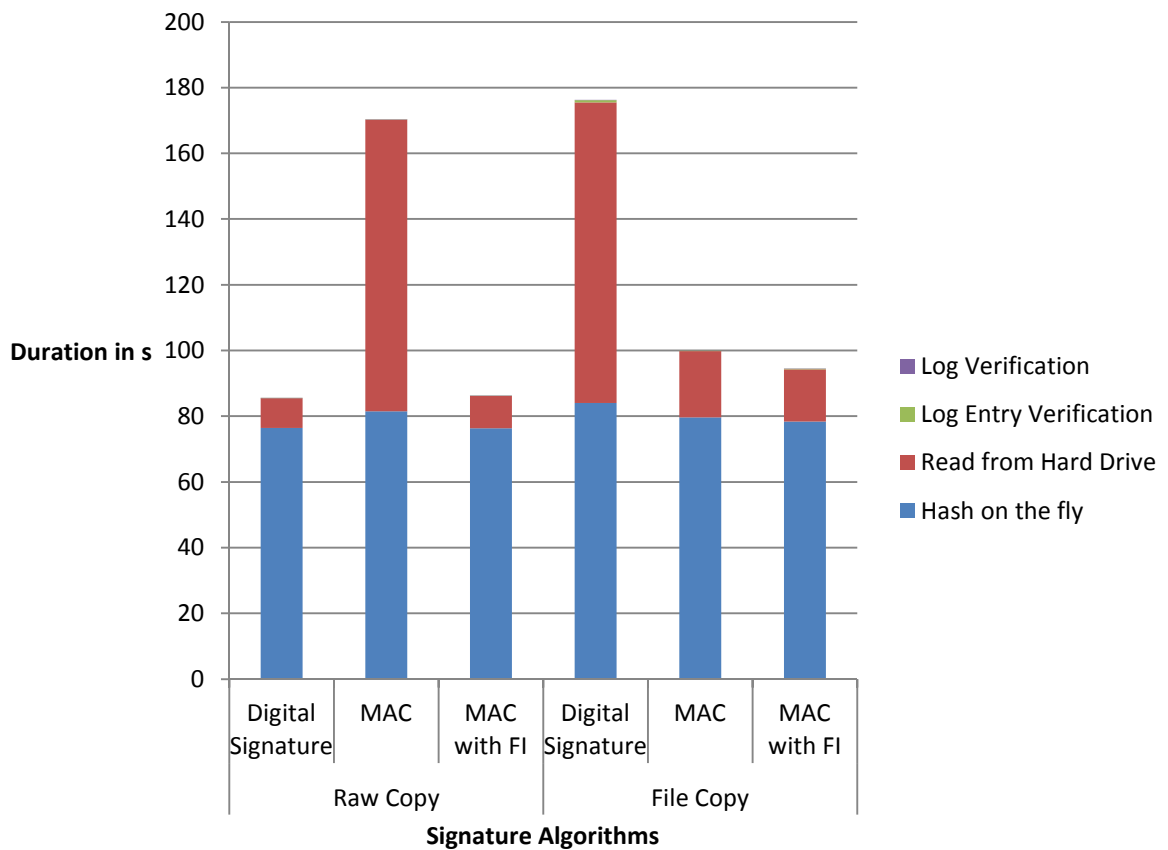


Figure 7.5: Medion Verification Test Results

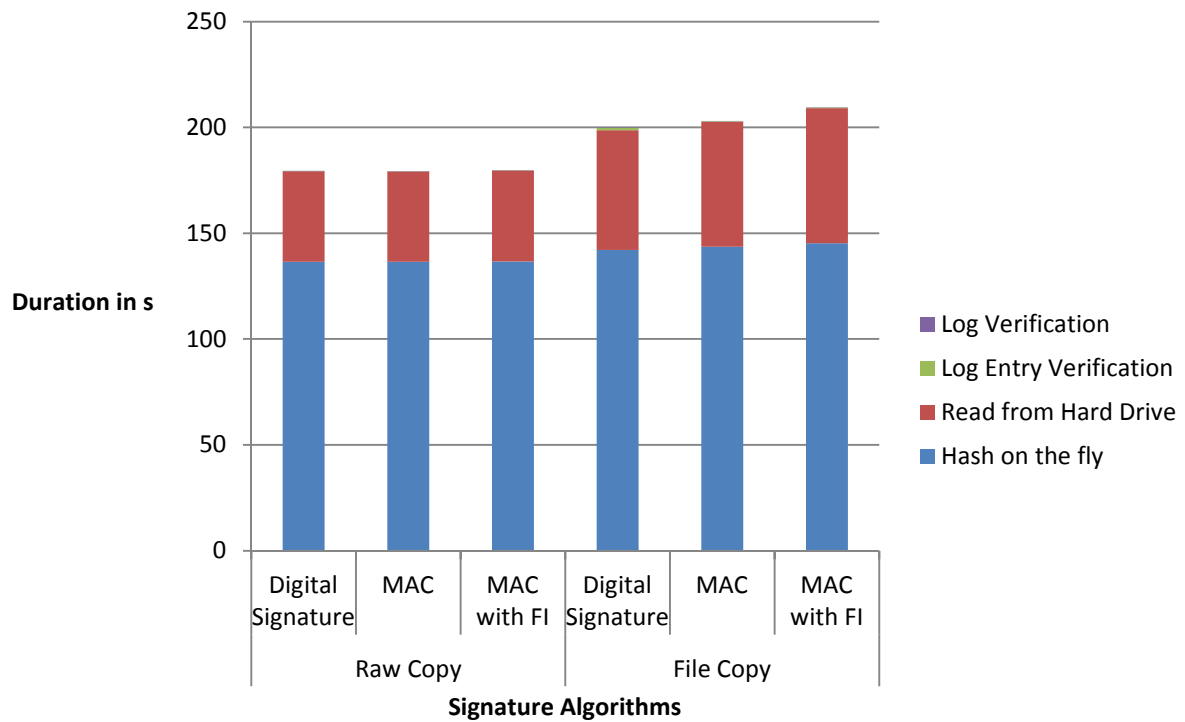


Figure 7.6: Dell Verification Test Results

The main finding can be summarized as follows:

- The hash calculations are the most time consuming and vary around 79 s for the faster CPU and 140 s for the slower one.
- The next biggest time consumer is reading from the hard drive with approximately 51 s for the Dell and 39 s for the Medion. Tests carried out on the Dell showed that the time needed to read is highly dependent on the fragmentation of the disc. The Dell test shown in this document was conducted using an external disc drive with more than 100GB of free capacity. Therefore, the Dell results show a small standard deviation only.
- The Medion results were achieved by copying to the local disc. In all the cases, the higher standard deviation of the read function (versus the Dell tests) was caused by one single test showing abnormally high read times. In many cases, the read operation from the hard drive was extremely fast: more than 4 GB were read in less than 10 s corresponding to a transfer rate of more than 400 MB/s. This only can be achieved by reading out of the disc drive cache.
- The entry and log verification is negligible in all cases. Although digital signatures in file copy mode consume much more time (due to the large number of files - 1700), in all cases these steps consume less than 1s.

8 Summary

In the following the results of this work are summarized. The outlook provides an overview of what still needs to be done to develop a production ready system.

8.1 Results Achieved

The results of this project demonstrate that secure logging with digital signatures enhances the functionality of the Omega Pro copying machine in away that the copied data provides digital evidence.

The logging mechanisms which have been proposed guarantee that the security requirements data integrity, log integrity, and authenticity are met.

Test results indicate that secure logging with digital signature adds approximately 28 % to 35% of costs in terms of run time. This overhead is predominantly due to the hash on-the-fly calculations made during the copying process. Log signing plays a role that can be neglected.

Data copied by the Omega Pro is admissible and credible. As a consequence, the productivity of the authorities using the copying machine is increased by using the efficient analysis processes that have been proposed. The analysis of optical media for digital evidence is no longer a necessity.

8.2 Prototype Boundaries

The prototype developed does not guard against all attacks prior to finishing the copying process. Examples: manipulating DVDs prior to copying (insert/replace) and malware on the copying machine. These threats must be addressed by organizational/technical means.

Hard drive failures on the external drive can potentially result in the verification process not working anymore. To guard against this, a raid protected external hard drive should be used.

8.3 Outlook

The prototype has been developed to prove that secure logging methods add value to the current copying process at an affordable cost. It is not production ready.

To enhance security, and to make the prototype is ready for production, the following enhancements must be put in place:

- The code has to be cleaned up by removing the timer functionality and the symmetric cryptographic classes.
- The Omega Pro has to be set up as a trusted server, able to communicate with the verifier in a secure way.

-
- The validation tool has to be enhanced to communicate with the trusted server.
 - The Omega Pro is able to copy several optical disc media in parallel to each other. The copying processes have to run in threads in order to implement this functionality. The prototype has been developed on a single thread basis only. Therefore, the prototype functionality needs to be enhanced by multi-threading the copying process. As long as one log per optical media is used, the logging process can remain as it is. Logs generated during this process should then be brought together. The combined log must be signed again.

9 Appendix

9.1 References

1. **Kahn Consulting Inc.** Computer Security Log Files as Evidence. [Online] August 2006. [Cited: May 10, 2011.] http://www.kahnconsultinginc.com/images/pdfs/KCI_ArcSight_ESM_Evaluation.pdf.
2. **Schneier, Bruce and Kelsey, John.** Secure Audit Logs to Support Computer Forensics. [Online] 1999. [Cited: May 15, 2011.] <http://www.schneier.com/paper-auditlogs.pdf>.
3. **Ma, Di and Tsudik, Gene.** A new Approach to Secure Logging. [Online] March 2009. [Cited: May 09, 2011.] <http://doi.acm.org/10.1145/1502777.1502779>.
4. **Bellare, Mihir and Yee, Bennet.** Forward Integrity For Secure Audit Logs. [Online] November 23, 1997. [Cited: May 10, 2011.] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.28.7970&rep=rep1&type=pdf>.
5. **Accorsi, Rafael.** Log Data as Digital Evidence: What Secure Logging Protocols Have to Offer? [Online] 2009. [Cited: May 10, 2011.] <http://www.informatik.uni-freiburg.de/~accorsi/papers/imf09.pdf>.
6. **Eckert, Prof.Dr. Claudia.** *IT-Sicherheit*. München : Oldenburg Verlag, 2005.
7. **Buchmann, Prof.Dr. Johannes.** *Einführung in die Kryptographie*. s.l. : Springer Verlag, 2008. Vol. 4.
8. **National Institute of Standards and Technology.** FIPS Pub 180-1: Secure Hash Standard. [Online] April 17, 1995. [Cited: July 08, 2011.] <http://www.itl.nist.gov/fipspubs/fip180-1.htm>.
9. **MIT Laboratory for Computer Science and RSA Data Security, Inc.** RFC 1321, The MD5 Message-Digest Algorithm. [Online] April 1992. [Cited: July 09, 2011.] <http://tools.ietf.org/html/rfc1321>.
10. **Wang , Xiaoyun and Yu, Hongbo .** How to Break MD5 and Other Hash Functions. [Online] 2005. [Cited: July 04, 2011.] <http://www.infosec.sdu.edu.cn/uploadfile/papers/HowtoBreakMD5andOtherHashFunctions.pdf>.
11. **National Institute of Standards and Technology.** FIPS Pub 197: Advanced Encryption Standard. [Online] 2001. [Cited: July 08, 2011.] <http://csrc.nist.gov/publications/fips/fips197/fips-197.pdf>.
12. **Bellare, Mihir , Canettiy , Ran and Krawczyk, Hugo .** Keying Hash Functions for Message Authentication. [Online] June 1996. [Cited: July 12, 2011.] <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.134.8430&rank=1>.
13. **Holt, Jason E.** Logcrypt: Forward Security and Public Verification for Secure Audit Logs. [Online] 2006. [Cited: May 10, 2011.] <http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.95.3741&rep=rep1&type=pdf>.
14. **Ma , Di and Tsudik, Gene.** Forward-Secure Sequential Aggregate Authentication. [Online] 2007. [Cited: May 08, 2011.] <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.78.7253&rank=1>.
15. **Lonvick, C.** RFC 3164: The BSD syslog Protocol. [Online] August 2001. [Cited: May 09, 2011.] http://datatracker.ietf.org/doc/rfc3164/?include_text=1.
16. **New, D. and Rose, M.** RFC 3195: Reliable Delivery for Syslog. [Online] November 2001. [Cited: May 15, 2011.] <http://datatracker.ietf.org/doc/rfc3195/>.
17. **Kelsey et al.** RFC 5848: Signed Syslog Messages. [Online] May 2010. [Cited: May 09, 2011.] http://datatracker.ietf.org/doc/rfc5848/?include_text=1.
18. **Waters, Brent R., et al., et al.** Building an Encrypted and Seachable Audit Log. [Online] [Cited: May 16, 2011.] <http://citeseer.ist.psu.edu/viewdoc/summary?doi=10.1.1.3.2936>.
19. **Ohtaki, Yasuhiro.** Partial Disclosure of Searchable Encrypted Data with Support for Boolean Queries. [Online] [Cited: May 20, 2011.] <http://portal.acm.org/citation.cfm?id=1371960>.
20. **Patterson, David, Gibson, Garth and Katz, Randy.** A Case for Redundant Arrays of Inexpensive Disks (RAID). [Online] 1988. [Cited: July 20, 2011.] <http://www-2.cs.cmu.edu/~garth/RAIDpaper/Patterson88.pdf>.

9.2 Performance Measurement Tables

Copying Measurements

PC	Medion
Function	Copy

	Dig. Sign.	Raw Copy		File Copy			Average
		MAC	MAC with FI	Dig. Sign.	MAC	MAC with FI	
Current Mode							
Read	256.390	266.179	263.976	280.902	270.382	269.079	267.818
Write	7.056	7.042	6.969	7.505	6.975	7.033	7.097
Subtotal	263.446	273.221	270.944	288.407	277.357	276.112	274.915
New Functionality							
Hash	92.861	94.028	91.527	93.237	92.135	93.361	92.858
SignEntr	36	1	1	17.395	350	345	3.021
SignLog	17	1	1	63	37	37	26
Subtotal	92.915	94.029	91.529	110.695	92.522	93.743	95.905
Total	356.362	367.249	362.473	399.102	369.879	369.855	370.820

PC	Dell EXT
Function	Copy

	Dig. Sign.	Raw Copy		File Copy			Average
		MAC	MAC with FI	Dig. Sign.	MAC	MAC with FI	
Current Mode							
Read	419.383	419.839	460.258	1.424.182	1.417.568	1.430.384	928.602
Write	12.202	11.683	11.542	11.877	12.747	12.670	12.120
Subtotal	431.585	431.522	471.801	1.436.059	1.430.315	1.443.054	940.723
New Functionality							
Hash	263.668	261.374	262.119	257.029	257.205	257.239	259.772
SignEntry	603	1	2	33.822	550	653	5.939
SignLog	42	1	1	107	91	86	55
Subtotal	264.313	261.376	262.123	290.958	257.845	257.979	265.766
Total	695.897	692.899	733.923	1.727.017	1.688.160	1.701.033	1.206.488

Verifying Measurements

PC	Medion
Function	Verify

	Dig. Sign.	Raw Copy		Dig. Sign.	File Copy		Average
		MAC	MAC with FI		MAC	MAC with FI	
New Functionality							
Hash	76.458	81.541	76.381	84.060	79.636	78.344	79.403
Read	9.065	88.740	9.820	91.432	20.138	15.929	39.187
VerifyEntry	1	1	1	697	209	214	187
VerifyLog	1	1	1	28	28	30	15
Total	85.525	170.282	86.202	176.217	100.011	94.516	118.792

PC	Dell EXT
Function	Verify

	Dig. Sign.	Raw Copy		Dig. Sign.	File Copy		Average
		MAC	MAC with FI		MAC	MAC with FI	
New Functionality							
Hash	136.490	136.447	136.588	142.111	143.694	145.264	140.099
Read	42.753	42.732	43.065	56.539	58.975	63.908	51.329
VerifyEntry	2	1	1	1.139	240	240	271
VerifyLog	3	1	1	37	41	45	21
Total	179.248	179.181	179.656	199.826	202.951	209.458	191.720