A Framework for Secure and Verifiable Logging in Public Communication Networks

Vassilios Stathopoulos[1] *, Panayiotis Kotzanikolaou[1] * and Emmanouil Magkos[2]

[1] Authority for the Assurance of Communications Security and Privacy (ADAE)
3 Ierou Lochou str., 15124 Maroussi, Greece
{v.stathopoulos,p.kotzanikolaou}@adae.gr
[2] Ionian University, Department of Informatics,
Palaia Anaktora, 49100, Corfu, Greece, emagos@ionio.gr

Abstract. In this paper we are focusing on secure logging for public network providers. We review existing security threat models against system logging and we extend these to a new threat model especially suited in the environment of telecommunication network providers. We also propose a framework for secure logging in public communication networks as well as realistic implementations designs, which are more resilient to the identified security threats. A key role to the proposed framework is given to an independent Regulatory Authority, which is responsible to verify the integrity of the log files.

1 Introduction

Public network providers, (fixed, mobile telephony and Internet Providers) consider privacy in communication as a valuable asset. Indeed, attacks against the confidentiality of communications and the privacy of their customers may lead to severe consequences of commercial and legal nature. In many countries Regulatory Authorities (*RA*s) are responsible to regulate and audit the security level of public network providers, in order to preserve communications security and privacy for the citizens.

Although a lot of security measures are in place in telecommunication networks and well defined standards exist there are still security holes. Threats such as external intrusion, communication interception, unauthorized access to private data (*e.g.* CDR files) and abuse of privileges by insiders must be considered. Existing vulnerabilities such as overestimation of security measures, non conformance with security measures and lack of dependable and secure logging and auditing mechanisms increase the security risks. Since it is not always possible to prevent security breaches, it is required to have in place adequate detective security measures.

* Research supported by the Hellenic Authority for the Assurance of Communications Security and Privacy (ADAE) – http://www.adae.gr.

System logging is the most important detective security measure. Log files are maintained in almost every system and they are usually examined during security audits, either external or internal. Indeed, during regular security audits, log files may be examined and correlated, in order to assure that the intended technical measures are in place and that the security policies and procedures are implemented. During non-scheduled security audits, *e.g.* as a response to a security incident, log files are analyzed in order to discover the cause of the incident, such as lack of security measures, non conformance with security procedures, system miss-configuration etc.

In this paper we are focusing on secure logging for public network providers. We review existing security threat models against system logging and we extend these to a new threat model especially suited in the environment of telecommunication network providers. We also propose a framework for secure logging in public communication networks as well as realistic implementations designs, which are more resilient to the identified security threats. A key role to the proposed framework is given to an independent Regulatory Authority. Each provider is responsible to send integrity proofs of its log files to the Regulatory Authority, which in turn is responsible to to remotely store the integrity proofs and verify the integrity of the log files.

Our paper is motivated from the recently announced interception case in a mobile telecommunications provider in Greece (see for example [1]). As the Greek authorities and the provider itself revealed, part of the core network of the provider was compromised by some unknown trojan-like program. According to published information, the malicious software infected the core network. Then, it activated the Lawful Interception (LI) component in the infected elements, which is by default installed in inactive mode, and made possible the call interception of several subscribers.[1] The malicious program turned off several logging procedures in order not to alarm about its presence or the fact that the LI component had been activated. The underestimation of several security threats and vulnerabilities regarding logging procedures and mechanisms, did not allow the immediate detection of the incident.

The rest of this paper is organized as follows. In Section 2 we review the related work in secure logging. In Section 3 we describe our threat model for secure logging in telecommunication networks in comparison with existing threat models. In Section 4 we describe the proposed framework for secure logging which deals with the identified threats. Finally, Section 5 concludes this paper.

2 Related Work

In *real logging systems*, the security of logging and auditing procedures is usually relied on the assumption that the host's Operating System is not corrupted. Secure systems aim at improving the robustness of the logging system itself without relying on the security features of the underlying system. The *Syslog-sign*

[1] The announced list of the victims included among others the Prime Minister, Ministers and Ex-Ministers.

IETF draft [2] describes a logging system with message source authentication and message integrity, built above *Syslog*, a cross-platform standard for remote logging on a central repository. In the ReVirt system [3] the OS is moved to a virtual machine and the integrity of the logging system is protected against external attacks with OS-level privileges.

Cryptographic research in secure logging systems aims at building logs that are irrevocably *tamper-evident*. In a scheme presented by Bellare and Yee [4], the MAC key that authenticates the logs entries is sequentially computed using an one-way cryptographic function, in order to achieve the *forward integrity* (FI) property, *i.e.* if an adversary compromises the current MAC key, she cannot modify old entries without being detected.

Schneier and Kelsey [5] propose a secure logging scheme that detects any attempts to delete or modify log entries on a host that has been compromised. Log entries are linked using a technique called *hash chaining* [6], where each entry contains a cryptographic digest of the previous entry. Moreover, each entry is encrypted and authenticated using an "epoch" secret that is updated using an one-way cryptographic function. The initial secret is shared with an external trusted party T who is able to independently verify the integrity of the logged data. The scheme of [5] satisfies the FI property and it allows the selective disclosure of the encrypted log data, using permission masks as an authorization list.

In recent works [7–10], the scheme of [5] has been extended to provide for keyword searching on the encrypted data using public key based cryptographic techniques [8], to enable tamper-evident remote logging in resource-poor devices [9], or to detect software tampering in DRM systems [7]. In [10] the *LogCrypt* system implements the scheme of [5] and extends it to support public-key signatures for accountability and public verifiability of the submitted logs, while it also discusses secure aggregation of multiple logs in a distributed logging system. Secure aggregation of multiple logs for forensic computing was also addressed in a recent scheme that uses distributed *Merkle Trees* [11] for the collection of the log files.

3 Threat Models

In the general case, the logs are generated by one or more log *Generators* (devices, systems, software e.t.c.) and are sent to the *Log Server* through a relay mechanism. Existing threat models include:

Trusted Generators and Marginally Trusted Log Server. In this model (*e.g.* [5]) the logs are generated and relayed to the Log Server within a trusted environment. However, although the log server is protected, it cannot be guaranteed that it will not be compromised. Consequently, in this threat model the security attacks which are mainly considered are disclosure and modification attacks against the stored logs.

Distributed Log Generators and Marginally Trusted Log Server. In this case (*e.g.*[9]) the log generators and the log servers are in a fully distributed en-

vironment. The log servers are considered, as in the previous case, marginally trusted. In addition to the previous model, attacks during the transmission of log data are considered. Since the logs are generated in a distributed environment, the log messages are not assumed by default to originate from the claimed device. Hence, attacks against the transmission of log entries are also examined such as: impersonation attacks against log generators, and impersonation and disclosure attacks against log messages during transmission.

3.1 Our threat model for public network providers

Existing threat models do not consider *insider attacks* and *collusion attacks* between log generators and log servers. Our threat model integrates and extends existing threat models in order to protect log files from attacks which have been identified within the environment of public network providers. For example, a possible threat may be that the log generators deliberately send modified log messages, or that the stored logs are deliberately modified after their storage to the log server with the active participation of the log server administrator.

Our threat model assumes that all entities involved in the logging process are *semi-trusted*, including the generator(s) of log entries, the log server(s) that stores the log files and the communication channel. Consequently, in addition to the threats described in the previous models, we also consider the following threats:

- Modification attacks on the stored logs from compromised log servers.
- Modification attacks from compromised log generators.
- Modification attacks from colluding log generators and log servers.

4 A Framework for Secure Logging in Public Networks

We consider a semi-trusted environment for the provider as described in Section 3.1. Thus, the security framework must protect logging systems from both external and internal attacks. We assume the existence of a trusted Regulatory Authority *RA* which is responsible to assure that the Providers take all reasonable measures to preserve communications' security and privacy. In regular audits or after a security incident, the *RA* may examine the log files of the provider, in order to determine the cause of the incident.

Provided that log files are important evidence, a security framework is required that will guarantee the availability and the integrity of logging operations and log files. Such a framework consists of the following phases:

4.1 Phase 1: Define the network and operational events

Before any security measures are taken, it is important to explicitly define what is important to be logged. This decision involves both the Provider and the *RA*. From the Provider's side, an effective logging supports system maintenance,

troubleshooting and internal security audits. From the *RA*'s side, logging information helps in investigating the cause of a security incident and as evidence in a court of law.

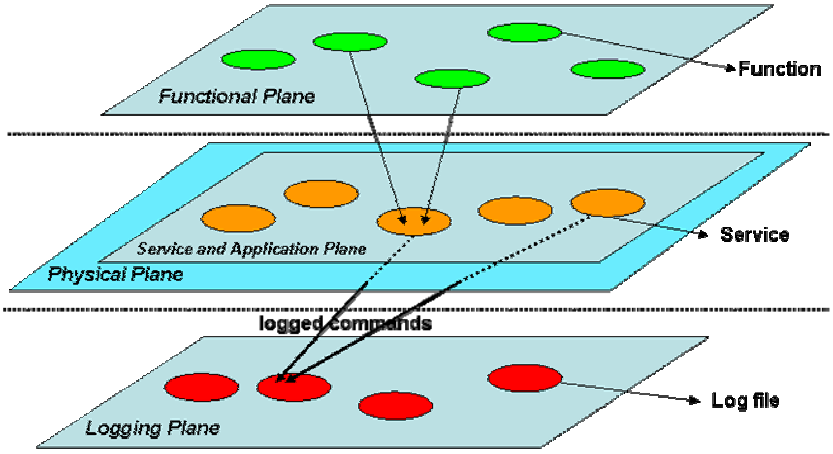

Fig. 1. An abstract representation of a Log Reference Model

In order to determine the events that must be logged within a Provider, a *Log Reference Model* is defined – see Fig. 1. This model is an abstract representation linking *Functions i.e.* general categories of network and operational events, to the corresponding *Log Files* that monitor these Functions, through the *Services* which implement the Functions. This model analyzes the logging needs from three different views, called *Planes*. These planes are:

- **Functional Plane.** It models the network and operational events within a network, without taking into consideration implementation details, architectural or topology constraints and design requirements. Suggestively and not limitedly in a provider's environment the following categories of Functions should be logged: (1) *Security Functions* (*e.g.* system access control, password management, user management, Lawful Interception, Data Retention). (2) *Service Management Functions* (*e.g.* monitoring, troubleshooting, management services) and (3) *Network Management Functions* (*e.g.* network configuration, network connectivity, routing).
- **Service and Application Plane.** It describes all specific services which are executed within the network or IT nodes. It discriminates system from application services, while it takes into consideration the OS platform, communication protocols and interconnections and hardware. Examples of services of this plane are the snmp service, the dsl service, the password management service, the AAA service, the radius service etc.
- **Logging Plane.** It describes specific commands and events of each used service, which can be grouped into separated log files. For example, the

command "show user" (captured for displaying user names) will be logged in a log file named "password management". This log file will correspond to the password management service, which implements part of the security management functions.

4.2 Phase 2: Define the requirements of log files

After the list of Functions to be logged has been interpreted to Services and consequently to Log Files, the operational and security requirements of each log file must be determined. In particular, requirements in this category define for each log file:

1. Log File Structure (the fields contained in each log file),
2. Required generation frequency,
3. Storage requirements (form, local and/or remote storage and storage duration).

Organizing logging and log files requirements requires the agreement among the Providers and the Regulatory Authority. This constitutes an administrative procedure without requiring any extra equipment at the providers' premises. Hence the cost is minimized.

4.3 Phase 3: Security measures against external attacks

In order to secure the log files from external and common internal threats all the functions which have been identified in the previous phases must be securely logged. This can be achieved by using the secure logging scheme of Schneier and Kesley [5] (see Fig. 2). We briefly describe this approach.

Each log server is supplied with an initial symmetric key A_0. The log file consists of consequent log entries $L_1, ..., L_n$. The key A_0 is updated for each new log entry, through a cryptographic one-way hash function $hash$, *i.e.* $A_i = hash(A_{i-1})$. Each log data entry L_i contains the log data D_i, which is encrypted and integrity protected. In order to encrypt the log data, a key K_i is derived from A_i, by hashing the concatenation of the key A_i with the permission mask W_i of the data entry D_i. Thus, $K_i = hash(W_i, A_i)$ and the encryption is $E_{K_i}(D_i)$. For integrity protection of the log entries, a hash-chain is used. Each log entry L_i contains the hash value $Y_i = hash(Y_{i-1}, E_{K_i}(D_i), W_i)$, (for Y_0 a padding value is used), as well as the Message Authentication Code $Z_i = MAC_{A_i}(Y_i)$. Thus, each time only one MAC and one hash value is stored, which contains all the previously hashed results. This preserves forward integrity from outsiders, since if the key A_i is not compromised, the attacker cannot modify the log entries undetected.

4.4 Phase 4: Security measures against internal attacks

Although after each log entry L_i is stored in the log file and A_i has been updated to A_{i+1}, the previous key A_i is deleted, it is possible for a compromised log

server to modify the log file. Suppose that the system is compromised at the time t_i, with or without the active participation of the log server administrator. This means that the current key, say A_i, is revealed to the adversary and also that the adversary has access to the log files from the time t_i and after. The adversary does not change the log entries at that time. Then, at time t_j, $j > i$, the adversary modifies the log entries $i, i+1, ..., j$. By using the key A_i that the adversary possesses, the keys $A_{i+1}, ..., A_j$ are reconstructed and the original log entries are replaced by the manipulated log entries. This attack cannot be traced, even if the logging mechanism simply replaces the MACs with digital signatures (as in [9]), since if the symmetric key is compromised, then all the keying material will have been compromised, including signature keys.

To deal with these attacks, we enhance the secure logging system described above with digital signatures and a trusted RA. In addition with the security measures described in the previous section, the following security measures are combined:

- Limited interaction of the provider with the trusted RA.
- Digital signatures of log files in predefined time periods.
- Digital signatures of log files in random time intervals.
- Remote storage of digital signatures in the RA.

The proposed extension is shown in Fig. 2. We assume that a secure communication interface is always available between the Provider and the RA. Moreover, in addition with the symmetric keys used to protect the log entries, each log server is assigned with two independent public/secret key pairs, PK_1/SK_1, PK_2/SK_2 and the corresponding digital certificates $Cert_1$, $Cert_2$. The digital certificates are issued by a trusted certification authority, so that all the parties can verify the validity of the signatures generated with the keys SK_1, SK_2. The key management functions such as generation, certification, revocation and updating of the signature keys may be supported by one or more independent certification authorities, which are trusted by the RA and the Providers.

Manual signing of the log files. In order to retain integrity proofs for the provider's log files, the RA periodically receives digital signatures of each log file. The RA has defined a signature period T for each log file[2]. The predefined signatures are generated as follows:

The Log Server administrator manually takes a copy of the log file instance and signs it in an isolated (off-line) environment. The signature is generated with the key SK_1. Note that SK_1 is not installed in the Log Server so that a successful attack in the Log Server will not affect the security of the key SK_1. Then, each signature $sig_1, sig_2, ...$ is send to the RA through a well-defined interface (marked as A in Fig. 2). The whole procedure is periodically repeated at the end of each period T.

[2] This can be performed in the second phase where the requirements of each log file are set

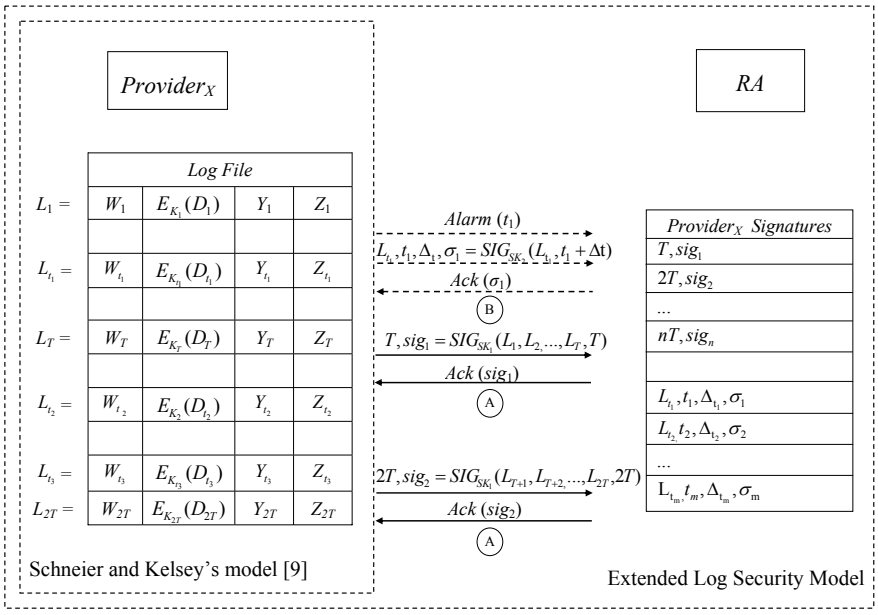

Fig. 2. Extending the secure log system

On receiving a signature, the RA acknowledges it and stores it in a secure storage space for future audits. These signatures can be later verified by the RA by accessing the certificate $Cert_1$ and the actual log file at the provider's premises. Since the RA holds the signatures of the log file instances, any malicious modification of these instances *after* the corresponding signatures have been sent to the RA can be detected, even if the symmetric keys $A_0, A_1, ...$ at the Provider's side have been compromised.

Since the predefined signing procedure is off-line and requires human intervention, it cannot be performed in short time periods. On the other hand, a very long time period T reduces the integrity protection of the protected log files. According to the security needs, it is recommended that T is between one day to one week. In order to reduce the operational overhead involved with the manual signing and the key management procedures, these can be integrated with ordinary manual operations of the administrators. Well-defined signing and key management procedures may reduce the additional operational costs to an acceptable level.

Automated signing of critical log events. As described above, the manual signing of the log files may detect any modification of a log file instance *after* the signature has been send to the RA. However, it does not protect from modification attacks, which have taken place *before* the signature has been generated.

In order to protect from modification attacks within signing periods, the *critical* log events are automatically signed and send to the RA. The automated signatures are generated as follows:

The RA defines a list of critical events, *i.e.* log events which should be immediately stored in the RA's side. A critical log event concerns actions which might be part of a malicious attack. Examples of critical events may include system restart, service mode modification (start, stop), modification of users and user privileges, modification of the log file and modification of the criticality level of a command.

Each critical event generates an *alarm*. A distributed *Alarm Service* is responsible to recognize the alarms and open a session with the RA. The alarm service has access to the signature key SK_2. It reads the critical log entry along with a number of the following log entries for a limited period Δ_t, signs them with the key SK_2 and sends the corresponding signatures $\sigma_1, \sigma_2, ...$ to the RA (marked as B in Fig. 2). At the end of the session, the RA acknowledges the receipt of the signature and stores the log event signature for future audits.

For additional security the RA can also request at random time intervals, signatures of the running log entries of the log file. A random request is processed as an alarm and may be performed several times within a period T, so that the RA has integrity evidence within a logging period.

Note that the alarm and the random request procedure is executed transparently from the Log Server administrator. Since the alarm service will sign any critical log event, it will be feasible for the RA to trace possible modification of log entries. Moreover, the use of random requests will increase the uncertainty of an attacker since the time of a random request will not be known to an attacker, even if the attacker is an internal user with advanced access privileges.

The automated signing procedure requires the development of dedicated software, such as a distributed execution environment for critical log event management and signing mechanisms. Before the implementation of this procedure a detailed cost analysis is required.

4.5 Phase 5: Implementation design

The above requirements need to be incorporated within an implementation environment. Figure 3 proposes a generic implementation design.

The implementation of security measures against external attacks requires the establishment of a synergy protocol between the RA and the Provider. Hence, from the Provider's side a *Mediation Device* is required with two interfaces; one for the communication with the RA and a second one for the communication with the internal entities. In the RA's side a *Switching Device* manages the corresponding interfaces.

The Mediation Device sends configuration commands to various *Physical Entities* (PE) (*e.g.* Remote Log Servers, Local Log Servers), receives captured information, collects this information and delivers this to the RA. The Mediation Device manages the automated signing procedure through the *Alarm Manager*.

The *Alarm Agents* operating within the Provider's physical entities are programmed to inform the Alarm Manager for the critical log events. The Alarm Manager has access to the key SK_2 and executes the automated signing procedure. It also forwards to the RA the off-line signatures of the log file instances, generated in an isolated Secure Signature Device, which stores the key SK_1. Finally, it receives the random requests for automated signing from a *Signature Request Manager* (SRM) and processes them as an alarm. Remote or local logging can be properly parameterized.

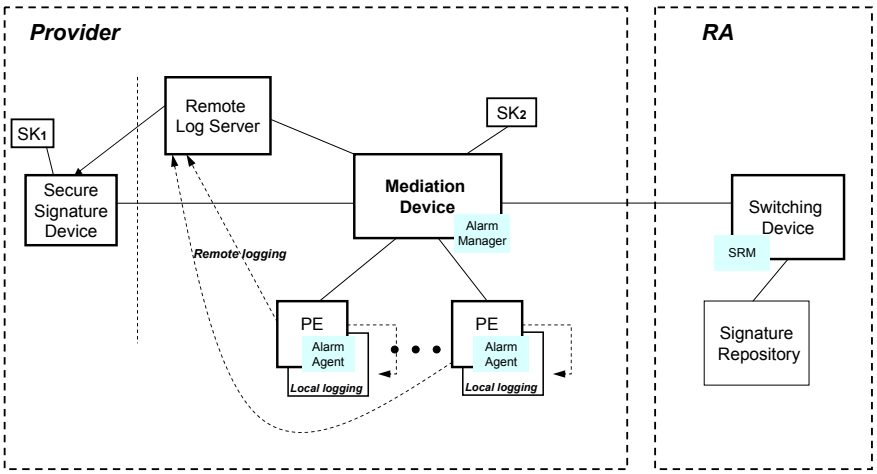

Fig. 3. An generic implementation design of logging management procedures

4.6 Phase 6: Log verification procedure

In case of a security audit or after a security incident, the RA will compare the log files that are stored in its environment, with the log files stored in the environment of the provider.

First, the RA will verify the validity of the signatures generated with the key SK_1. Recall that in the previous phase, the log administrator uses this key off-line in order to periodically sign the instances of the log file. The verification requires access to the public key PK_1, derived from $Cert_1$ and to the log file itself, stored into the Log Server. If some signature is not verified, then the RA has evidence that the log file stored in the Provider's side has been modified.

If all the deterministic signatures of the log file are verified, then the RA will verify the randomly generated signatures. Again, this requires access to the certificate $Cert_2$ and to the specific log entries of the log file. If all these signatures are verified, then the RA accepts the validity of the log file. In case one or more signatures are not valid, then the RA has strong indication that

some of the log entries have been modified, before the entire log file was signed by the administrator. This also provides valuable information about the time of the intrusion to the log server.

5 Security Analysis

The use of the secure logging scheme of [5], protects the confidentiality and the integrity of the log files from external and common internal threats. Indeed, an attacker cannot read the log files or modify them without being detected, provided that the key A_0 or the keys derived from it, are not revealed.

The use of manual off-line signatures, protects the log files from modifications by external or even internal attackers, $after$ the file has been signed and the signatures have been sent to the RA. Indeed, if a malicious or compromised log server administrator attempts to modify the log files by using the symmetric key A_0, after the file has been signed for the current period, the modified log file will not match with the corresponding signature, which is remotely stored within the RA. Recall that the manual signing procedure is performed off-line and thus even if the log server has been compromised by an intruder, it will not be possible to generate a valid signature for the compromised log file. Moreover, the use of manual signatures of the log files also provide non-repudiation for the Provider, since in case of a security incident, the Provider will not be able to repudiate a signature of the log file.

In addition, the automated signing of the critical log events and the remote storage of these signatures, minimizes the available time-frame for an intruder to manipulate the log events, before their storage to the log file and the manual signing of the log file. Indeed, even if all the systems of the Provider have been compromised, including the physical entities (switches, routers etc), the Mediation Device, the local log files and the log servers, it will be extremely hard to prevent all the alarm agents from generating and sending an alarm to the RA.

Even if an internal attacker could be able to intercept the automated signatures and attempt to modify their values before they are sent to the RA, it would require that the Alarm Manager and all the Alarm Agents in all possible affected systems have been properly compromised. Even if the attacker controls the majority of the Provider's systems, he cannot be assured that his attack will not be logged. Note that the Alarm Manager and the Alarm Agents are software entities that have been approved by the RA (for example by using singed code). Any possible attempt to update their code, or modify them will also cause a critical event and the generation of an automated signed event towards the RA.

6 Conclusions

Existing secure logging systems mainly protect the log files from external attacks. In public communication networks however, the security requirements of log files must also consider internal attacks such as compromised log generators, compromised log servers or combinations of both. In this paper we consider an

extended threat model for logging systems and we define a generic framework for secure logging for public network providers. Through the proposed framework the logging requirements of each provider are defined, as well as the required security measures and procedures for the protection of the log files. A trusted Regulator Authority RA has a central role in this framework, in the definition of the logging requirements as well as in the integrity verification of the maintained log files. In addition with known security measures for secure logging, we propose the use of digital signatures in two different ways, as well as the remote storage of the signatures in the environment of the RA. Although modification attacks against log files cannot always be prevented, it is possible to detect such attacks with well defined mechanisms and procedures.

References

1. Schneier, B.: Schneier on security: Phone tapping in Greece. web page, http://www.schneier.com/blog/archives/2006/02/phone_tapping_i.html (2006)
2. Kelsey, J., J.Callas: Ssyslog-sign protocol. DRAFT, Network Working Group (2002)
3. Dunlap, G.W., King, S.T., Cinar, S., Basrai, M., Chen, P.M.: Revirt: Enabling intrusion analysis through virtual-machine logging and replay. In: In Proc. 2002 Symp. Operating Sys. Design and Implementation. (2002)
4. Bellare, M., Yee, B.: Forward integrity for secure audit logs. Technical report, Computer Science and Engineering Department, University of California at San Diego (1997)
5. Schneier, B., Kelsey, J.: Cryptographic support for secure logs on untrusted machines. In: Proceedings of the 7th USENIX Security Symposium, USENIX Press (1998) 53–62
6. Haber, S., Stornetta, W.: How to time-stamp a digital document. In Menezes, A., Vanstone, S.A., eds.: Proc. of CRYPTO'90. Volume 537 of Lecture Notes in Computer Science., Springer-Verlag (1990) 437–455
7. Chong, C.N., Peng, Z., Hartel, P.H.: Secure audit logging with tamperresistant hardware. Tech. Rep., Universiteit Twente, Enschede, The Netherlands (2002)
8. Waters, B., Balfanz, D., Durfee, G., Smetters, D.: Building an encrypted and searchable audit log. In: The 11th Annual Network and Distributed System Security Symposium. (2004)
9. Accorsi, R.: On the relationship of privacy and secure remote logging in dynamic systems. In: Security and Privacy in Dynamic Environments. Volume 201., Springer-Verlag (2006) 329–338
10. Holt, J.: Logcrypt: Forward security and public verification for secure audit logs. In: Proc. of Australasian Information Security Workshop. (2006)
11. Kawaguchi, N., Obata, N., Ueda, S., Azuma, Y., Shigeno, H., Okada, K.: Efficient log authentication for forensic computing. In: In Proc Of IEEE 6th Information Assurance Workshop, IEEE (2005) 215–223