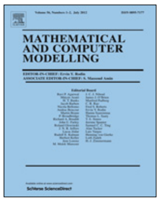
Contents lists available at SciVerse ScienceDirect
Mathematical and Computer Modelling
journal homepage: www.elsevier.com/locate/mcm



A secure log architecture to support remote auditing

Rafael Accorsi
Business Process Security Group, University of Freiburg, Germany

ARTICLE INFO

Article history:
Received 27 May 2011
Received in revised form 21 June 2012
Accepted 27 June 2012

Keywords:
Secure log architecture
Public key cryptography
Remote auditing
Secure digital archiving

ABSTRACT

This paper presents BBox, a digital black box to provide for authentic archiving (and, consequently, forensic evidence) for remote auditing in distributed systems. Based upon public key cryptography and trusted computing platforms, the BBox employs standard primitives to ensure the authenticity of records during the transmission from devices to the collector, as well as during their storage on the collector and keyword retrieval by authorized auditors.

1. Introduction

The rapidly growing number of national and international compliance requirements emphasizes the importance of archiving of business processes transactions, where records are analyzed as part of an audit to corroborate or refute potential violations of compliance rules [1]. To this end, authentic records are essential to guarantee reliable accountability and non-repudiation of actions [2].

This is particularly relevant in the context of remote auditing (RA) [3]. Teeter et al. define RA as the process by which auditors couple information and communication technology with data analytics to access and report on the accuracy of financial data and internal controls, independent of the physical location of the auditor [4]. RA thus eliminates the location constraint of an audit and is inherently connected with some form of networking to connect the entity to be audited (so-called *auditee*) and the auditor.

By outsourcing computation, e.g. onto cloud or grid computing, both internal and external auditors do not have access to the physical systems in the cloud. In particular, external auditors will most likely have to access both the auditee's system and the auditee's compartment in the systems in order to conduct the examination. In this case, the auditors may employ RA to examine both the auditee and the system provider. This clearly motivates the use development of RA as a means to make audits possible and to provide high assurance for clouds, thereby addressing their inherent loss of control [5].

While this makes evident the demand for digital black boxes as a means to guarantee the authenticity of records, to-date this is realized with ordinary logging services that record log file entries for the events happening in the system. These services alone are not sufficient for sound authenticity guarantees [6]. Although several proposals for secure logging services exist, none of them ensures sufficient authenticity guarantees for both log data in transit and at rest and, at the same time, allow the selective disclosure of records to auditors [7].

Contributions. This paper presents BBox, a digital black box to provide for authentic and confidential system records in distributed systems. Authenticity comprises different forms of *integrity* of log events and records in their various phases (see Section 2 for details). *Confidentiality* is not a strictly mandatory security requirement for all scenarios, but it is often a non-functional property required for sensitive data. In fact, the importance of confidentiality stems largely from the context within which the BBox is employed. Generally, it is simply unreasonable to store entries in the clear, even though encryption

E-mail address: accorsi@iig.uni-freiburg.de.

0895-7177/$ – see front matter © 2012 Elsevier Ltd. All rights reserved.
doi:10.1016/j.mcm.2012.06.035

considerably complicates the log files search [8]. For these requirements, the BBox addresses the collection, the transmission, the storage and the retrieval of records, thereby providing:

- *Reliable data origin*. Only events sent by authorized devices are recorded in log files. Provenance information is stored in log entries for further investigation as a kind of hearsay statement, ensuring liability and accountability.
- *Tamper-evident storage*. Through the use of hash chains, log entries are stored in a way that tampering attempts, such as adding counterfeit entries or modifying the payload of legitimate entries, can be detected by a verifier.
- *Encrypted records*. Records are not stored in clear-text, but encrypted with a unique, evolving key, thereby providing for *forward secrecy*: if an attacker surreptitiously obtains the key of some entry, this attacker cannot deduce the keys used to encrypt the previous entries [9].
- *Keyword-based retrieval of records*. Despite the encryption of entries, the BBox allows for simple keyword searches for log entries, thereby generating the so-called ''log views''. In doing so, the retrieval solely requires the decryption of entries matching with the keyword. This not only reduces the cost of log view generation, it also enforces that only the necessary information is disclosed to auditors, acting thereby as an access control mechanism.

This paper presents the design of the BBox, which largely builds upon public key cryptography, secure communication protocols and trusted computing modules. Specifically, the architecture, key algorithms and protocols are presented and, according to a powerful attacker model, their properties are demonstrated. Besides this, we also report on a prototypical implementation of the BBox, as well as on its evaluation using state of the art vulnerability analysis methods [10,11] and concrete attack techniques and strategies.

Application and adjacent aspects. The current application of the BBox is in the setting of business process management [12]. Specifically, the BBox is a component of a business process management system (BPMS) to guarantee authentic archiving during the workflow execution in a service-oriented architecture and, hence, allow the forensic analysis [13–15] and reliable business process mining [16]. To this end, the workflow specification language is slightly extended with tags to define the service-side devices empowered to communicate events to the BBox (residing at the BPMS). This realization of the BBox builds upon standard protocols for web services communication and workflow execution, addressing the criteria in [17]. The remainder of this paper presents the high-level cryptographic building blocks, whereas the evaluation refers to the prototypical implementation as a standalone application. In doing this, the paper does not refer to the particular implementation of the BBox for BPMS. The paper also does not refer to the trusted computing modules and protocols involved in providing the security guarantees associated to the BBox. See [18] for details on this part.

Organization. Section 2 defines the basic terminology employed in the paper, the usual logging architecture and the attacker models for secure logging. Section 3 builds the core of the paper, presenting the architecture, components and operation of the BBox. Section 4 addresses the retrieval of log views. Section 5 reports on the security analysis of the BBox, demonstrating its main properties. Related work is discussed in Section 6. Section 7 discusses further research topics for digital black boxes.

2. Logging and attacker models

This section defines the terminology and security requirements in the log setting and reports on the related work, emphasizing the gaps addressed by the BBox.

2.1. Terminology and security requirements

We assume the architecture proposed in [19]. It consists of subjects that may play one of three roles: *devices* capture events and send them to *collectors* that store the events in log-files. (*Relays* between the devices and the collectors may exist, but are often omitted.) Authorized *auditors* retrieve the collector and obtain the relevant portions of the log file (i.e. *execution traces*) for a RA. Log messages sent by devices to the collector are *in transit* and messages recorded in the log file are *at rest*. Parts of the log file retrieved by auditors are *in processing*. Two additional roles may exist: $\mathcal{V}$ is a semi-trusted verifier and $\mathcal{T}$ is a trusted verifier. These principals assert the authenticity of and administrate the secrets used to protect the audit trail.

Log files are either *sequential* or *circular*. The former grow indefinitely as entries are appended. For the latter, there is a bound – e.g. the total number of log entries or file size – that whenever reached, forces the oldest entries to be overwritten with the new ones. In the context of digital evidence, and so the BBox, log files are taken as being sequential. In settings where the storage is limited (e.g. smart-meters and ubiquitous computing [20]), the local collectors operate in a circular fashion, while the remote collectors use sequential files.

We categorize security requirements pertinent to secure logging according to the state of log data addressed by the BBox, namely in transit and at rest.

Log data in transit. Log *messages* are in transit between the device and the collector, where the following requirements apply:

- *origin authentication*: the collector must ensure that log messages were sent by authorized devices.
- *message confidentiality*: the log messages must remain confidential during the transmission.
- *message integrity*: the message cannot be modified during the transmission.

- *message uniqueness*: log messages are logged just once.
- *reliable delivery*: log messages sent by the device must reach the collector. This is not related to denial of service (DoS) attacks, but with the transport protocol employed.

These requirements must also hold when the collector communicates with $\mathcal{V}$ and $\mathcal{T}$.

Log data at rest. In the storage phase, the collector appends log *entries* to the audit trail. The following requirements are set:

- *entry accountability*: log entries must include information with regard to the subject (i.e. device and collector) that appends the entry to the audit trail.
- *entry integrity*: audit trails cannot be changed once recorded. Specifically, we define integrity as "accuracy" (log entries were not modified), "completeness" (log entries were not deleted), and "compactness" (log entries were not illegally appended).
- *entry confidentiality*: entries are not stored in clear-text.

Tamper evidence is needed while enforcing some of these requirements. If an attacker succeeds in violating the integrity, e.g. by altering the contents of a log entry, an auditor must be able to detect this tampering.

The security requirements concerning log data in transit and at rest are necessary, yet not sufficient. Complementary, non-architectural measures protecting, e.g., the access to log files and the isolation of the logging processes to avoid covert-channels [21] are necessary as well. Similarly, devices could be subverted – thereby sending semantically wrong messages – or the messages sent by the devices could be removed from the network (denial of services). The consideration of such attacks traditionally falls outside the scope of secure logging protocols.

2.2. Attacker models

Each of the aforementioned phases has a corresponding attacker model. Note that both attacker models assume that attackers do not possess the secrets used in the logging protocol, i.e. the cryptographic keys are initially *not* compromised. If an attacker gains access to these secrets, no security properties can be guaranteed.

Attacker model for the log data in transit. This phase considers an attacker residing in the network and attacking log messages between the device and the collector over a relay, and those between the collector and $\mathcal{V}$ and $\mathcal{T}$. The attacker model for this setting is that of Dolev–Yao [22]. The attacker controls the communication channel in the sense that he can insert, eavesdrop, delay, schedule, modify, and completely block transmitted messages. However, the attacker can only gain access to the contents of an encrypted message if he possesses the corresponding keys; and he can only produce meaningful encrypted messages if the attacker possesses the necessary sub-messages. The attacker is also computationally bounded, being subject to the same complexity constraints for cryptographic computation as legitimate principals.

Attacker model for log data at rest. This phase considers a penetrator inside the collector attacking directly log data at rest in the audit trail. Such an attacker can read, (over)write, (re)move and delete (fields of the) log entries. In doing so, such an attacker may generate message items from the items he already possesses. However, as in the case of log data in transit, the attacker can only obtain the contents of an encrypted log entry if he possesses the corresponding decryption key. It also assumes that the attacker is computationally bounded with regard to cryptography.

General considerations. As mentioned above, devices could be attacked and successfully impersonated, so that an attacker could inject messages which do not correspond to the real events happening in the systems. In fact, because the devices denote the weakest link in the security chain, this is one of the most frequent attacks in secure logging in enterprise information systems [10,11]. More importantly, they are impossible to discover. That is because the entries themselves are, from the viewpoint of security, authentic; only their semantics does not faithfully reflect the events in the system. This, of course, tricks the (remote) auditors into believing they possess an "authentic" log, thereby leading to wrong audit reports. The only means to detect such an attack is to test the device in recurrent time-frames, and change the keys to avoid compromise.

Another aspect worth noting regards the capabilities of an auditor. While we refrain from considering this as an attacker in the strict sense, the auditor could employ queries to obtain more information than necessary, or chain queries in way that the inference of important attributes is possible. This is a relevant problem in general and is usually coupled to entry (or region) level access control. In the following, we consider only the keyword search as an access control mechanism. The BBox assumes an authorization mechanism in that only authorized auditors are allowed to query the log. Furthermore, accountability is provided, as the BBox stores all the queries in the operational log file.

3. BBox: architecture and logging algorithms

Fig. 1 depicts the high-level architecture of the BBox. While the BBox cannot check the veracity of the events, i.e. whether these events really correspond to what happens in the system, it ensures that only authorized devices submit *log messages* and that no subject other than the BBox accesses these messages. This is achieved with public key cryptography. Accredited

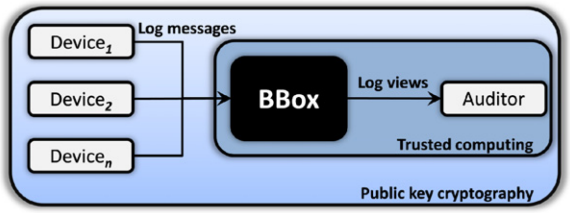


Fig. 1. The high-level architecture of the BBox logging.

auditors may query the BBox to obtain *log views*, i.e. audit trails containing the all log entries matching the search criteria.[1] Here, a protocol ensures the mutual authentication between collector and auditors, and a trusted computing platform, in particular remote attestation, ensures that the corresponding secure logging protocols are in place.

3.1. BBox architecture components

The architecture of the BBox, its components and the information flows happening therein are depicted in Fig. 2. Its functionality can be distinguished in two logical units: the "recording unit" is an input channel for logging communicated events in a secure manner and consists of the log message and the entry append handlers; the "retrieval unit" is an output channel for the generation of log views which encompasses the log view and the entry retrieval handlers. In detail:

- *Log message handler* (LMH). The LMH receives incoming log messages sent by the device and carries out an integrity check to determine (1) whether the contents of the message are eligible to be appended to the secure log file and (2) whether the devices' certificate are legitimate.
- *Entry append handler* (EAH). If a log message passes the integrity test carried out in the LMH, its payload and keyword ID are given to the EAH, which transforms these components in a protected entry for inclusion into the secure log file. To this end, it employs the protocol described in Section 3.3.
- *Secure log file*. This is the container where events are securely recorded after being prepared by the EAH.
- *Log view handler* (LVH). The LVH controls the disclosure of collected data by receiving view requests, authenticating auditors and passing on the necessary information to the entry retrieval handler.
- *Entry retrieval handler* (ERH). The ERH receives the keyword ID of the requesting individual and produces a corresponding query over the secure log file. To this end, information in the crypto module is needed in order to allow the decryption of the corresponding log entries.
- *Crypto module*. This is a trusted computing module (TPM) responsible for, among others, storing the cryptographic keys, providing meta-data and a basis for remote attestation.
- *Operational log file*. The functioning of the BBox is recorded in a write-once, read-many operational log file. Events recorded in this file include, e.g., the decision whether a log message has passed the integrity test and service disruptions, such as (re-)initializations and shutdowns.

Notation. The presentation of the BBox employs the following notation: d_i denotes the ith device; P_i refers to the *payload* of the ith log message; K_s stands for the *public key* of a subject s and K_s^{-1} stands for the corresponding *private key*; K_i with $i \in \mathbb{N}$ stands for a *symmetric key* and the *symmetric encryption* of X with K_i is denoted by $\{X\}_{K_i}$; $\{X\}_{K_s}$ denotes the *asymmetric encryption* of message X under the key K_s. $\{X\}_{K_s^{-1}}$ stands for the *signature* of X by s with K_s^{-1}; $Hash(X)$ stands for the *one way hash* of X; X, X' denotes the *juxtaposition* of X and X'; and E_i stands for the ith *log entry*.

The BBox assumes that the cryptographic primitives exhibit the expected properties, e.g., it is infeasible for an attacker to intentionally cause collisions of hash values or calculate the pre-image of hash functions, and that decryption of messages requires the appropriate cryptographic key. A further assumption is that no subject other than s possesses his private key K_s^{-1}.

3.2. BBox initialization and incoming log messages

Assuming that the BBox is not compromised and initially offline, its initialization phase encompasses four steps. The first step places the asymmetric key pair K_{BBox} and K_{BBox}^{-1} into the crypto module and synchronizes its internal clock with a reliable clock. (These keys are *not* the same as the attestation key of the crypto module.) Based on these keys, the second step generates the value $G_0 = Hash(K_{\mathrm{BBox}}^{-1})$ which is used as basis for the secure logging service (see Section 3.3).

The third step at initializing the BBox appends the device authorization and key lookup table (DAKL) to the LMH. This table is necessary to authenticate devices, as they must have been previously authorized to send messages to the BBox.

[1] The concrete shape of the keyword depends on the application, e.g. it could be the ID of a particular subject to which the record refers.

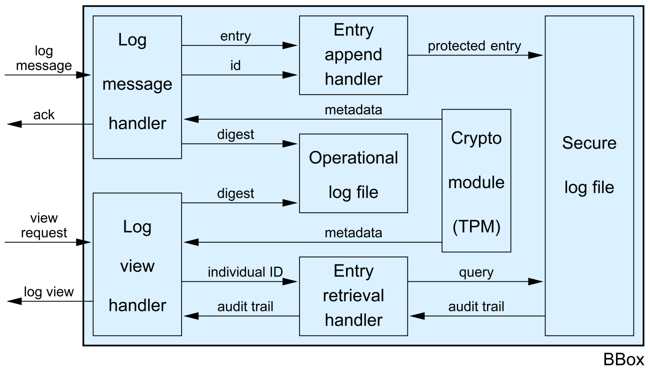


Fig. 2. BBox: Architecture, components and main information flows.

Table 1
Excerpt of a BBox' DAKL table.

Devices' MAC	Public key	Comment
`00:C0:9F:30:A6:1B`	`9TPYHfeWH+Bok5rgMa...`	`RFID reader #43`
`00:F0:4A:23:B2:AA`	`OuanFjE7W4vjo6KLly...`	`RFID reader #12`
`00:0B:6A:04:97:20`	`whbUE3xfIC+JafigeI...`	`Database server #1`

As depicted in Table 1, each entry in the DAKL table contains the identifier of a device expressed by its MAC address, the respective public key and a human readable comment about the particular device. Thus, adding devices to the system on an already online BBox leads to an update of the DAKL and a corresponding entry in the operational log file.

The fourth and final step consists in opening both the *secure* and the *operational* log files and appending the corresponding initialization entry to them.

Receiving log messages. The communication between the device and the BBox does not require mutual authentication. It only prescribes that events are encrypted using the public key of BBox and signed by a legitimate device. In detail, the log message communicating the event P sent by a device d_i carrying a keyword I to the BBox at time t is denoted as

$$(\textit{Log message})\ d_i \rightarrow \text{BBox} : \{d_i, \{I, P, t\}_{K_{d_i}^{-1}}\}_{K_{\text{BBox}}},$$

where the identity d_i of the device is expressed in terms of its MAC address.

Upon the receipt of a log message the LMH carries out an integrity check whose goal is to assert that: (a) the device d_i is allowed to communicate events to the BBox; (b) the message has not been altered along the way between d_i and the BBox; (c) the received message is not a replay of an expired log message.

The integrity test consists of the following steps. First, BBox decrypts the message using its private key K_{BBox}^{-1} and uses the DAKL table to check whether the sending device d_i is legitimate and, if so, whether its certificate has not been revoked. Second, the signature, and thereby the integrity of the message's payload, is verified. Third, if the checksum is validated, the time-stamp is checked to avoid replay attacks. If it has not expired, the event P and the keyword I are passed to the EAH for inclusion in the log file. Messages that fail to comply with the integrity requirements are discarded and the corresponding entry is included in the operational log file.

3.3. Appending log entries

The core of the EAH is a secure logging protocol. Parameterized by the keyword and payload of an entry, its goal is to generate a secure log entry by applying a series of cryptographic primitives to them and append it to the secure log file.

Cryptographic building blocks. The cryptographic building blocks used to generate the entries are the *evolving cryptographic key* G used to compute K (see Section 3.2) and the links of the *hash chain*.

In contrast to usual cryptography, where keys are kept the same over time, in evolving key cryptosystems keys change, or evolve, from time to time, thereby limiting the damage that can result if an attacker learns the current cryptographic key [23]. In the BBox, each payload is encrypted with a unique key K_i derived from an evolving *entry authentication key* G_i and the entry identifier I by hashing these two values. The entry authentication key G evolves for each entry. Hence, the keys $K_i, K_{i+1}, \ldots$ are independent from each other, so that if the attacker obtains a particular K_i, he can neither obtain K_{i-1} nor K_{i+1}. To make it harder for attackers to decrypt the payload of messages, the entry keyword I is stored as a hash value, so that even if the attacker obtains some G, he still has to obtain I to gain access to the payload.

A hash chain is a successive application of a cryptographic hash function to a string [24]. By knowing the initial value and the parameters with which the chain is generated, the integrity of an existing hash chain can be checked for broken links by

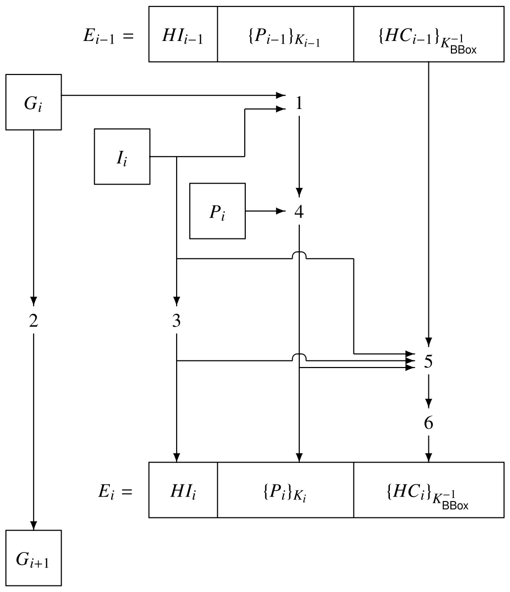


Fig. 3. Appending an entry to the secure log file.

recomputing each element of the chain. (Alternatively, it is also possible to check the integrity of contiguous regions of the chain instead of its whole.) The BBox uses hash chains to create an interdependency between an entry *i* and its predecessor $i - 1$, thereby linking entries to each other. Moreover, since elements of the hash chain can as well be seen as a checksum of the involved parameters, in computing an element of the chain and comparing it with the existing link, the BBox can also assert whether the corresponding entry has been modified or not. Hence, tamper evidence for integrity properties are accounted for.

Format of the entries and secure logging protocol. Fig. 3 depicts the format of the entries E_{i-1} and the steps used to produce the subsequent entry E_i. Each entry consists of the hashed keyword (denoted *HI*), the payload encrypted under the unique symmetric key (denoted $\{P_i\}_{K_i}$) and the signed link of the hash chain (denoted $\{HC_i\}_{K_{\mathsf{BBox}}-1}$).

Assuming that the *i*th log message has passed the integrity test, the EAH receives the keyword I_i and the payload P_i from the LMH and, with the value G_i at hand, performs the following operations:

1. $K_i = Hash(G_i, I_i)$ generates the cryptographic key for the *i*th log entry.
2. $G_{i+1} = Hash(G_i)$ generates the authentication key for the entry E_{i+1}.
3. $HI_i = Hash(I_i)$ computes the hash of the keyword of the entry E_i.
4. $\{P_i\}_{K_i}$ is payload P_i encrypted with K_i.
5. $HC_i = Hash(I_i, HI_i, \{P_i\}_{K_i}, \{HC_{i-1}\}_{K^{-1}_{\mathsf{BBox}}})$ is the *i*th link of the hash chain.
6. $\{HC_i\}_{K_{\mathsf{BBox}}-1}$ is the signed hash chain value for the *i*th entry.

These operations are depicted as a diagram in Fig. 3, where the numbers labeling the arrows correspond to those in the description of the operation. The labels also encode the order in which the operations are carried out. The resultant log entry, denoted $E_i = (HI_i, \{P_i\}_{K_i}, \{HC_i\}_{K^{-1}_{\mathsf{BBox}}})$, consists of the index HI_i, the encrypted log entry $\{P_i\}_{K_i}$ and the signed hash chain value HC_i. The authentication key G_{i+1} is employed to append the next incoming entry.

3.4. Authentication of secure log files

Authenticating secure log files means making tampering evident to a verifier, so that corrective measures can be taken to repair the file, e.g. using a rollback. The BBox employs the hash chain to this end: intermittently or before generating a log view, the secure log file can be authenticated, thereby excluding certain forms of tampering attempts. This is achieved by Algorithm 1, which roughly speaking "traverses" the hash-chain seeking for broken links.

Algorithm 1 requires a hash table *Hash_Table* relating the values *HI* with the corresponding pre-image values *I* necessary to compute the links of the hash chain, the file handler/pointer for the secure log file *LogFile* and the initial entry authentication key G_0 and the current entry authentication key $G_{current}$. It returns the variables *Integrity* and *Result*: *Integrity* is a Boolean variable, assuming True if no tampering has been detected in *LogFile* and False otherwise; *Result* encodes the

Algorithm 1 Log File Authentication

```
Require: $Hash\_Table$, $LogFile$, $G_0$, $G_{current}$
Provide: $Result$, $Integrity$
 1: $G \leftarrow G_0$
 2: if not EMPTY($LogFile$) and CHECK-FIRST-ENTRY($LogFile$) then
 3:     $Integrity \leftarrow$ True; $Result \leftarrow 0$
 4: else
 5:     $Integrity \leftarrow$ False; $Result \leftarrow 1$
 6: end if
 7: while ($Integrity$) and not EOF($LogFile$) do
 8:     $G \leftarrow$ HASH($G$)
 9:     if CHECK-LINK-SIGNATURE($Entry.HC$) then
10:         $HC \leftarrow$ DECRYPT($Entry.HC$ using $K^{-1}_{\mathsf{BBox}}$)
11:         if $Entry.HI \in Hash\_Table$ then
12:             $I \leftarrow$ LOOKUP-PRE-IMAGE($Hash\_Table$, $Entry.HI$)
13:             if $HC \neq$ HASH($I$, $HI$, $Entry.Payload$, $PreviousEntry.HC$) then
14:                 $Integrity \leftarrow$ False; $Result \leftarrow 2$
15:             end if
16:         else
17:             $Integrity \leftarrow$ False; $Result \leftarrow 3$
18:         end if
19:     else
20:         $Integrity \leftarrow$ False; $Result \leftarrow 4$
21:     end if
22: end while
23: if ($Integrity$) and (HASH($G$) $\neq G_{current}$) then
24:     $Integrity \leftarrow$ False; $Result \leftarrow 5$
25: end if
```

kind of tampering found during the authentication, namely: 0 if no tampering is detected; 1 if the initial entry has been tampered with; 2 if the hash chain is broken, i.e. the expected and actual values of a hash chain link diverge; 3 if the hashed index value is not listed in the *Hash_Table*; 4 if the signature of the hash chain link is invalid; and 5 if the length of the hash chain does not match with the expected length of *LogFile* (implicitly encoded by current entry authentication key $G_{current}$).

Algorithm 1 starts by checking the integrity of the initial entry. (This case must be singled out because E_0 exhibits a special format.) An empty log file indicates a tampering, as well as a flawed initial entry. If testing the initial entry succeeds, the remaining entries of the log file are tested while no tampering is found, i.e. $Integrity =$ True, and the end of file has not been reached (Line 7). This while-loop starts by evolving the entry authentication key (Line 8), overwriting the previous value with the new one. It then checks whether the signature of the ith link of the hash chain is valid (Line 9). If so, the algorithm checks whether the index is listed in the *Hash_Table* (Line 11) and, if this is the case, compares the actual value of the hash link with the computed value, i.e. the value obtained by recomputing the *HC* link (Line 13). If this test succeeds, the algorithm moves to the next entry. If one of these tests fail, the *Integrity* variable is set to False (Lines 14, 17 and 20) and the tampering attack is encoded in the variable *Result*.

If no tampering is detected during the while-loop, it traverses the whole log file. In this case, the length of the log file is tested, using to this end the entry authentication key G (Line 23): G is the result of the n iterations of the hash function, where n is the number of entries in the *LogFile*; $G_{current}$ is the result of $m + 1$ applications of the hash function, where m is the number of appended entries by the EAH. Ideally, $n + 1$ and $m + 1$ must be equal, otherwise the actual and the expected number of entries do not coincide, indicating tampering. Specifically, this indicates the deletion of entries (Lines 23 and 24).

4. Retrieval of log views

The concept of log view bears similarity with its homonymous counterpart in databases, where a view can be thought of as either a virtual table or a stored query, thereby acting as a filter on the underlying tables referenced in the view. Four components of the BBox are involved in the generation of log views. The LVH is responsible for performing the following two protocols.

Mutual authentication between the auditor and the BBox. To ensure that log view is generated for the pertinent auditor, entity authentication. From an auditor's viewpoint, he must also be aware that they communicated with the correct BBox. Thus, *mutual* authentication between requesting individuals and the BBox is necessary. The prototypical protocol used to achieve mutual authentication is a variant of the Needham–Schroeder public key protocol [25], which is slightly modified to comply with the prudent protocol engineering principles provided in [26]. The protocol is as follows:

Algorithm 2 Entry Selection

```
Require: $I$, $LogFile$, $OpLogFile$, $G_0$, $G_{current}$
Provide: $T$, Authentication_Failure
 1: $integrity \leftarrow$ True
 2: $keyword \leftarrow$ HASH($I$)
 3: $G \leftarrow G_0$
 4: while ($integrity$) and not EOF($LogFile$) do
 5:   $G \leftarrow$ HASH($G$)
 6:   if CHECK-ENTRY-INTEGRITY then
 7:     if $keyword = Entry.HI$ then
 8:       $K \leftarrow$ HASH($G, I$)
 9:       $P \leftarrow$ DECRYPT($Entry.Payload$ using $K$)
10:       APPEND-TO-BUFFER($T, P$)
11:     end if
12:   else
13:     $integrity \leftarrow$ False
14:   end if
15: end while
16: if ($integrity$) and (HASH(G) $\neq G_{current}$) then
17:   return $T$
18: else
19:   WRITE(Authentication_Failure in $OpLogFile$)
20:   return $T$, Authentication_Failure
21: end if
```

1. $A \rightarrow \mathsf{BBox} : \{A, N_A\}_{K_{\mathsf{BBox}}}$
2. $\mathsf{BBox} \rightarrow A : \{\mathsf{BBox}, N_A, N_{\mathsf{BBox}}\}_{K_A}$
3. $A \rightarrow \mathsf{BBox} : \{N_{\mathsf{BBox}}, \mathsf{BBox}\}_{K_{\mathsf{BBox}}}$

where N_x stands for the nonce of the principal x and A for the auditor.

Remote attestation of the BBox. Authentication provides no assurance with respect to the configuration of the BBox, e.g. whether the secure log mechanism is in place and the algorithm for retrieving log views is reliable. To achieve such guarantees, the LVH uses remote attestation protocols provided by the crypto module. The two-step protocol transfers the platform configuration registers (PCR) to the auditor. In detail:

1. $A \rightarrow \mathsf{BBox} : \{\mathtt{Att\_Req}, A\}_{K_{\mathsf{BBox}}}$
2. $\mathsf{BBox} \rightarrow A : \{\{PCR_{\mathsf{BBox}}\}_{AIK_{\mathsf{BBox}^{-1}}}, \mathsf{BBox}, t\}_{\mathsf{BBox}^{-1}}$

where AIK stands for the attestation identity key and t for the time-stamp.

If these two protocols run as expected, the ERH receives the keyword I and searches the secure log file for the matching entries. The resultant audit trail T contains all the entries related to I. T is given to the LVH, which is responsible for computing the metadata M (e.g. number of entries in T and generation time-stamp) and sending the resultant tuple T, M to the requesting individual.

Selection of log entries. Algorithm 2 shows how entries are selected and appended to the audit trail T. It consists of a linear search over the secure log file, where the integrity of the hash chain is checked when searching for the matching entries (Line 6). This integrity test consists of checking the signature of the hash chain link, decrypting its contents and checking whether the expected and actual contents correspond. If the integrity test fails, the search is aborted and the failure is recorded in the operational log file *OpLogFile* (Line 19) and reported to the BBox (Line 20). If a matching entry is found (Line 7), i.e. if the keyword searched matches with the keyword of the entry, the corresponding symmetric key K is computed (Line 8). With K, the payload of the entry is decrypted (Line 9) and appended to the buffer T (Line 10).

Provided that the log file has not been tampered with, the search finishes when all the entries of the secure log file are visited, producing the audit trail T. The LVH then encrypts the tuple T, M and sends it to the requesting auditor A.

5. Security analysis of the BBox

The cryptographic building blocks and protocols provide for authentic and confidential log data in transit and at rest. While the BBox provides for tamper evidence, its design does not account for tamper resistance. To this end, other measures, such as partial confinement, rollbacks and firewalls should be in place.

This section demonstrates the extent to which authenticity is achieved by the BBox and reports on experiments carried out with the prototypical implementation of the BBox. Along with the security analysis, this section discusses the main design decisions and assumptions underlying the BBox.

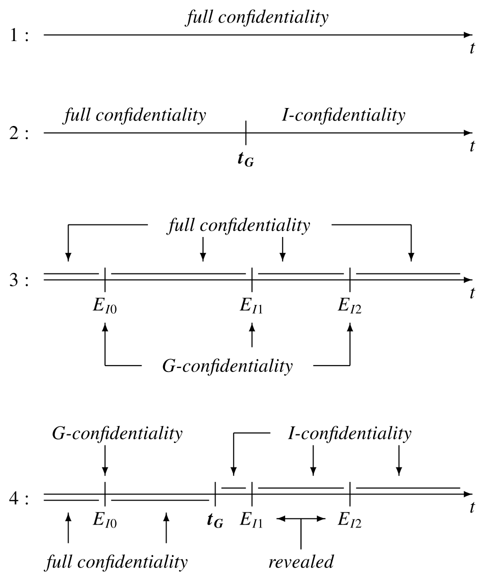


Fig. 4. Types confidentiality guarantees.

5.1. Log data in transit

Digital signatures are used to sign log messages sent from the devices to the BBox, thereby ensuring that only authorized devices, i.e. those listed in the DAKL, can submit log messages. The one-way authentication protocol (see Section 3.2) has been verified in the AVISPA tool for authentication and replay-freeness properties [27]. Specifically: the analysis aimed to check whether BBox could be tricked into accepting a log message from an illegal device, or a replayed message. Given a Dolev–Yao attacker model [22], no such attack could be found. Moreover, since the messages are encrypted, confidentiality against an eavesdropper is preserved.

The use of a time-stamp averts the possibility of replay attacks, provided the replay happens after the specified timespan. Although this timespan is kept short, it could still be exploited to store replicated messages. However, this would be inoffensive for the authenticity of the log file: the replay only leads to identical copies of a legitimate entry in the secure log file and auditors aware of this could simply filter the entries, keeping only their first appearance.

The use of a time-stamp could be circumvented if the device and the BBox mutually authenticate, guaranteeing the freshness and origin authentication of a message. However, this causes an overhead that cannot be justified in various application scenarios, e.g. pervasive and ubiquitous computing settings [20].

With regard to the communication between auditors requesting log views and the BBox, the prototypical protocol authenticating these peers is based on the Needham–Schroeder public key protocol. This protocol has also been formalized in the AVISPA tool and checked against man-in-the-middle attacks, where no attack could be pinpointed. The protocol for remote attestation is a standard TPM protocol. In particular, we employ the concept of "persistent link" to eliminate one relevant kind of attack in which a corrupted BBox could trick an auditor into accepting a log view generated by a flawed BBox algorithm [28].

5.2. Log data at rest

Two security properties are relevant for log data at rest: confidentiality and integrity. Below, we address these properties, demonstrating the extent to which they are fulfilled by the BBox.

Confidentiality properties. Confidentiality is achieved by encrypting the payload of the entries with the symmetric key K. This key is generated using the evolving key G (derived form the private key of the BBox) and the keyword for the entry I. Since I is not stored in clear in the BBox and G does not leave the TPM, log data at rest can be considered confidential. If an intruder – by guessing or inference – obtains the some key G_i for the ith entry, it is still impossible to decrypt the contents of the entries appended after E_i, as the intruder does not possess the keyword I. (This confidentiality guarantee is called "I-confidentiality".) Similarly, if the intruder gets aware of some I, all the entries prefixed with the hash of I could be decrypted upon the event of discovering the value G. (This confidentiality guarantee is called "G-confidentiality".) If the intruder knows both I_i and G_i for some entry E_i, then no confidentiality guarantee is provided.

These different confidentiality models are depicted in Fig. 4. Model 1 depicts the ideal situation in which for all the entries in the log file, the attacker possesses neither G nor I, so that all the entries fulfill full confidentiality. Model 2 shows the case where an attacker gains access to some entry authentication key G, thereby dividing the log file into entries that are fully confidential, i.e. those appended before t_G, and those I-confidential. The segregation is due to the evolution of G by means of hash functions: since none of the pre-images of G can be recomputed, for entries appended before t_G the attacker does not possess their G, nor the index I. Assuming that the attacker possesses some identifier I, Model 3 shows that for the entries with index I, namely the entries E_{I0}, E_{I1}, E_{I2}, only G-confidentiality is given, whereas the entries with indexes other than I still exhibit full confidentiality. Model 4 combines the previous models, showing the case in which the attacker possesses some index I and gains access to an entry authentication key G at time point t_G. As a result, the entries appended *after* t_G have either the release state – if they have the identifier I, as E_{I1}, E_{I2} do – or are I-confidential. For entries appended *before* t_G, either G-confidentiality (for entries having index I, such as E_{I0}) or full confidentiality holds (for entries having index other than I).

The confidentiality properties provided by the BBox rely on how difficult it is to gain access to both the I and the G values necessary to compute K. Since the value G does not leave the crypto module and since the values I are transmitted in the clear between the components of the BBox, it is more probable that I values become compromised than an attacker obtains some G. Hence, Model 3 is the more plausible, so that G-confidentiality holds for the entries with known indexes and full confidentiality holds for entries exhibiting indexes I' with $I' \neq I$.

If an attacker solely obtains some entry authentication key G, the cryptographic primitives based on evolving crypto systems employed in the logging protocol provide for *backward confidentiality*: the knowledge of a particular entry authentication key G does not compromise the previous keys G', in particular the master key G_0 and K^{-1}_{BBox}. Specifically, the (symmetric keys of) entries appended to the log file *before* the point in time when the attacker obtains the entry authentication key G exhibit some confidentiality guarantee (see Model 2).

The use of a hash table to relate HI with its pre-image I is a vulnerable spot of the BBox. If an attacker succeeds in obtaining this table, then not only the confidentiality of entries would be harmed, but also, e.g., the privacy of users. In [29], we employ the BBox to store the events of customers in a retailer using different forms of customer communication and ubiquitous computing. Here, I stands for the unique identifier of the customers, so that in the case of an attack, the events of the customers could be linked and their identities could be disclosed.

Integrity properties. With regard to integrity guarantees, Algorithm 1 is responsible for detecting attacks upon log entries, i.e. inclusion, modification and deletion of entries. To demonstrate its correctness, we consider an attacker model in which an intruder can read, (over)write, mode and delete (fields of the) log entries stored in the secure log file. In doing so, the attacker may generate message items from the items he already possesses. However, the attacker can only obtain the plain-text of an encrypted message if he possesses the corresponding decryption key. Moreover, it is also assumed that the attacker is computationally bounded.

Given this attacker, the correctness property of the Algorithm 1 is defined as the absence of false negatives: whenever the algorithm decides that a log is authentic, then there was no tampering—modification, appending and removal of entries.

Definition 1. Algorithm 1 is correct if an only if it does not exhibit false negatives. □

Theorem 1. *Algorithm* 1 *is correct with regard to authentication.* □

To show Theorem 1, every attack tampering attempt on an initially authentic log file must be examined, thereby demonstrating that they are detected.

Proof. Let *LogFile* be an authentic log file consisting of a sequence of entries $E_0, \ldots, E_n$ constructed according to Section 3.3, where each entry E_j, with $0 \leq j \leq n$, assumes the form $E_j = (HI_j, \{P_j\}_{K_j}, \{HC_j\}_{K^{-1}_{\text{BBox}}})$.

Case 1: *Modification attacks*. Let E_j be an entry of the *LogFile*. The modification of each of the three entry fields (i.e. entry's index, payload and hash chain link) must be analyzed in isolation.

(1.1) Overwriting the index of E_j: the attacker either overwrites the index HI_j of E_j with the index of an existing entry E_k or generates a new index H'_j. For the former case, the algorithm detects a broken hash chain by computing HC'_j and determining that $HC_j \neq HC'_j$ (Line 13). For the latter case, the value H'_j will not be contained in the *Hash_Table*, indicating the violation (Line 11).

(1.2) Overwriting the encrypted payload of E_j: irrespective of whether the attacker reuses the payload field of an existing entry E_k or generates a new payload field from the items he possesses, such an overwriting leads to a broken hash chain, as the expected value for the link of the hash chain diverges from the actual value (Line 13).

(1.3) Overwriting the hash chain link of E_j: the attacker either overwrites the link $\{HC_j\}_{K^{-1}_{\text{BBox}}}$ with the link of an existing entry E_k or generates a new link. For the former case, the algorithm detects a broken hash chain by computing HC'_j and determining that $HC_j \neq HC'_j$ (Line 13). For the latter case, since the attacker does not possess K^{-1}_{BBox}, he cannot generate a legitimate link that passes the signature check (Line 9).

Case 2: *Appending attacks*. Let E_j, E_{j+1} be contiguous entries of the *LogFile*.

(2.1) Appending at the beginning of *LogFile*: Let $j = 0$. Irrespective of whether the attacker copies an existing entry E_i, with $i > 0$ or generates a new entry E'_i, the initial entry will not correspond to the prescribed entry, so that checking the initial entry (Line 3) indicates a failure. For the special case in which the inserted is log entry E_0: while the check of the initial entry succeeds, the algorithm detects a broken hash chain (Line 16).

(2.2) Appending between E_j, E_{j+1}: The attacker either replicates an existing entry E_i or generates a new entry E_i', appending it between E_j and E_{j+1}. For the former case, the algorithm detects a broken hash chain, as the actual value of the hash link HC_i and the computed value HC_i' are not equal (Line 16). For the latter case, the attacker cannot produce a validly signed hash link (detected in Line 12).

(2.3) Appending after E_j, E_{j+1}: If $j \neq n$, i.e. E_{j+1} is not the last entry of *LogFile*, then Case 2.2 applies. Otherwise, the attacker can either replicate an existing entry or generate a new entry. For the former case, a broken hash chain arises (detected by Line 16). For the latter case, the signature of the hash link will not be valid (detected by Line 12).

Case 3: *Deletion attacks*. Let E_j, E_{j+1}, E_{j+2} be contiguous entries of the *LogFile*.

(3.1) Deleting E_j for $j = 0$: the check of initial entry (Line 3) will fail.

(3.2) Deleting E_{j+1} for $j > 0$: this leads to a broken hash chain, as the expected value HC will not correspond to the actual value HC_{j+1} (Line 16).

(3.3) Deleting E_{j+2} for $j = n - 2$: this is the last entry of *LogFile*. For this case, the entry authentication key G computed during the execution of the algorithm and the actual entry authentication key $G_{current}$ will diverge, indicating the attack (Line 29).

The case distinction shows that each tampering attack upon the integrity of the log data at rest is detected by the authentication algorithm, thereby demonstrating that Algorithm 1 is correct. □

5.3. Properties of log view generation

As log data at rest, the log views must be accurate, complete and compact. These properties are necessary, as if log views do not fulfill them, the result of auditing such log views cannot be considered correct.

Definition 2 (*Accuracy, Completeness and Compactness of Log Views*)**.** Let *LogFile* be a log file and I an identifier. A log view $L = T, M$ for I obtained from *LogFile* is *accurate* iff T contains the exact payloads of the log messages sent to the BBox; T is *complete* iff T encompasses the payloads of all the log messages received by the BBox related to I. L is *compact* iff T contains only the entries related to I. □

The generation of log views is done by Algorithm 2, which selects the entries from the log file according to the index I.

Theorem 2. *Algorithm* 2 *outputs accurate, complete and compact log views.* □

Accuracy, completeness and compactness of log views follow from tamper evidence upon the assumption that the attacker model is analogous to that of log data at rest, that the private key of the BBox is not known by the attacker and that the devices communicate every event occurring in the system.

Proof. Let *LogFile* be a log file consisting of a sequence of entries $E_0, \ldots, E_n$ constructed according to Section 3 and T be the audit-trail of the log view generated for I. T is constructed by linearly searching for entries E such that $Hash(I) = E.HI$ (see Line 8). Since the entry authentication test (Lines 7 and 17) detects tampering attempts (modification, insertion and deletion, see Theorem 1), log views are only generated if the source log file passes the authentication test. Specifically, this ensures that:

- no payload was modified, so that accuracy of log views is given.
- no index was modified or replaced, and no entry was deleted, so that all the entries for I are considered for selection (Line 8), providing for completeness.
- no entry was appended to *LogFile*, providing for compactness.

Hence, Algorithm 2 produces accurate, complete and compact log views. □

5.4. BBox prototype

The BBox has been realized as a prototype using the programming language Java and standard protocols for remote attestation found in trusted computing platforms TCB 1.1b. To obtain practical evidence as to whether the proposed algorithms and techniques provide the expected confidentiality and integrity guarantees, we employed ATLIST – a state of the art vulnerability analysis technique [11] – to identify potential vulnerabilities. Moreover, we conducted man-in-the-middle and tampering attacks to observe how the BBox behaves.

The vulnerability analysis with ATLIST pointed to three weak spots of the BBox: first, the possibility of impersonation attacks, both when attackers impersonate the devices and the auditors. Second, the hash-table linking the hash values to their pre-images. Third, the storage of the credentials used to create and assert the authenticity of the log file. Given that, we extensively tested the BBox for middle-man attacks and impersonation attacks, demonstrating that such attacks were not possible. The other two weak spots regard tamper-resistance and cannot be tackled with the algorithms implemented in the BBox; in fact, they fall outside the scope of the implementation as a whole.

Focusing on tamper evidence, we simulated every possible tampering combination of log files—considering the attacker model mentioned above. The goal was to demonstrate that these attack attempts are detected by the implementation of

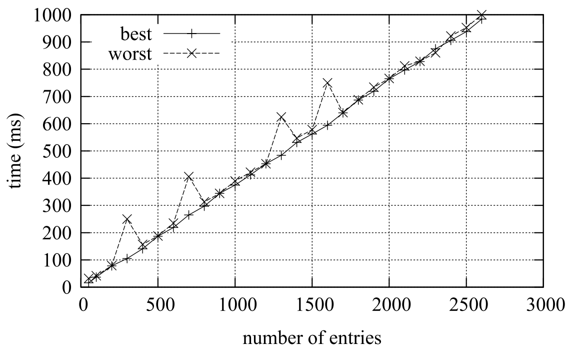


Fig. 5. Time necessary to authenticate a secure log file.

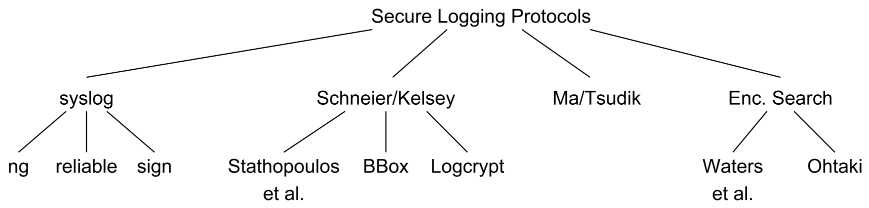


Fig. 6. Classes of secure logging protocols.

the algorithm. The BBox succeeded in authenticating the log files and detecting the attack attempts. For illustration, Fig. 5 depicts the runtime necessary to authenticate sample log files. (These runtime figures were obtained in a standard desktop PC with 1.92 GHz, 512 MB of RAM and 250 GB hard disk operating under MS-Windows and Java version 1.6; each entry payload amounts to 50 kb.)

While being based on a prototypical implementation, these runtime figures already indicate to one problem of secure logging implementation based on standard libraries: their performance is rather poor for use in large applications. Hence, besides improving the prototypical implementation, we currently investigate the benefits brought by fast hash chains verification techniques, e.g. [30].

Finally, denial of service attacks can be carried out upon the prototypical implementation of the BBox. We simulate these attacks by sending, over a period of five minutes, a constantly rising number of log messages (between 500 and 2500 messages per minute; step 500). Legitimate entries sent within the third minute were no longer added to the log file. While we currently investigate this issue in more detail, these preliminary figures already indicate that receiving messages is, in practice, not costly. In fact, the process of adding entries to the log file is rather expensive and turns out to be a bottleneck in the BBox architecture.

6. Related work

We group the existing logging protocols in four classes according to the techniques they build upon, as in Fig. 6. The “Syslog” class consists of protocols based on the syslog; the “Schneier–Kelsey” class consists of protocols extending that of [31]; the “Ma–Tsudik” class includes the novel logging techniques in [32]; the “Encrypted Search” class contains protocols to search encrypted log data.

Syslog class. Proposals in the former category focus on log data in transit. The “reliable-syslog” improves the transport protocol of syslog from UDP to TCP, thereby ensuring reliable message delivery [33]. The syslog-sign extends syslog with signature blocks preventing tampering of log data in transit [34]. However, since these signature blocks are loosely coupled to the entries and can be deleted after storage, protection for log data at rest is not given. Moreover, the entries are transmitted in the clear. Distributions of *nix systems have been equipped with syslog-ng, the successor of the syslog [35]. Besides reliable transfer over TCP, it also supports IPv6 and encrypted log message transfer using the TLS protocol.

Schneier–Kelsey class. Unlike the extensions of the syslog, proposals based on the Schneier–Kelsey scheme focus rather on log data *at rest*. Accorsi presents a simple extension of this scheme to address distributed storage [36]. Stathopoulos et al. present the application of Schneier–Kelsey’s scheme for the telecommunication setting [37] and Chong et al. employ it for DRM [38]. These proposals exhibit a similar vulnerability: the “tail-cut” attack differently reported in [7,39]. Logging services based on the Schneier–Kelsey scheme employ hash chains [24] to create dependencies between log entries. Hence, removing one or more log entries from the “chain” makes tampering detectable to verifiers. However, if the attacker removes entries from the end of the log file (i.e. the attacker truncates the log file), the verifier cannot detect the attack unless he is aware of the original length of the hash chain. While employing similar cryptographic primitives as Schneier–Kelsey, the BBox is not susceptible to tail-cut attacks.

Ma–Tsudik class. Ma and Tsudik [32] propose a logging protocol for the storage phase that employs a novel authentication technique called "Forward-Secure Sequential Aggregate" (FssAgg) [40].

In an FssAgg, forward-security, append only signatures generated by the same signer are sequentially combined into a single aggregate signature. Successful verification of an aggregate signature is equivalent to that of each component signature, whereas failed verification of an aggregate signature implies that at least one component signature is invalid. Ma and Tsudik use FssAgg as a means to achieve integrity, just as Schneier and Kelsey use hash chains. The use of FssAgg offers a number of technical advantages over hash chains, though. Besides *correctness* – i.e. any aggregated signature produced in the scheme can be verified –, a secure FssAgg scheme satisfies the *forward secure aggregate unforgeability*, i.e. no one including attackers knowing the current signing key can make a valid FssAgg forgery. The latter implies two properties: first, a secure signature is append-only; second, it is computationally infeasible to remove a component signature without knowing it. Carried over to the logging protocol, these properties allow Ma and Tsudik to build a secure logging protocol that provides *forward security* as Schneier and Kelsey [31], *stream security*, so that audit trail reordering is impossible, and *integrity*, i.e. insertion of new entries as well as modification and deletion of existing entries renders the final aggregated invalid.

Encrypted search class. Some settings assume a collector that records the entries in clear-text and, eventually, stores the resultant audit trail (or backups thereof) in marginally trusted sites. This makes it necessary to encrypt the *whole* audit trail. The downside of this measure is that encryption makes it extremely difficult to search for and extract entries from the audit trail, which is essential for digital evidence generation. In fact, a naïve approach to do so would require decrypting every entry separately. This brings along a number of disadvantages, such as unintended access to and possible disclosure of classified data, and huge computational cost associated with entry description.

This class of logging protocols is concerned with the secure storage of encrypted audit trails and efficient retrieval of entries from these trails. (See [41] for techniques tailored for encrypted search only.) We classify the existing protocols into those based on *Identity-Based Encryption* (IBE), i.e. a special kind of PKI where *any* string can be used as a public key [42], and those approaches based on *Bloom Filters*, i.e. probabilistic data structures to test whether an element is a member of a set [43]. (While false positives are possible in Bloom Filters, false negatives are not.)

Other secure logging services. Xu et al. propose the SAWS architecture, the secure audit web server [44]. Since no algorithms are provided, only the architectural similarities between the BBox and SAWS can be compared: both employ public key cryptography and trusted computing modules for storage, but transmission and retrieval capabilities are not addressed. Recently, Ma and Tsudik proposed a novel approach to secure logging based on cumulative signatures that replace the use of hash chains [32]. This approach also focuses on data storage and it is currently unclear whether it is susceptible to "tail-cut" attacks. It is, from the performance viewpoint, an approach that is clearly slower than the use of hash chains.

Compared with the aforementioned state of the art, the advantages of the BBox are the simultaneous provision of: first, authenticity protection for both data in motion and at rest, providing for tamper evidence guarantees. Second, keyword search in encrypted log files. Third, during an audit, attestation that the certified algorithms are running, thereby enhancing the probative force of evidence generated using the BBox [45].

7. Summary

The paper introduced the architecture and main components of the BBox, a digital black box to ensure authentic archival of records as a basis for remote auditability. The paper defines the corresponding algorithms and shows the extent to which they provide the authenticity (and confidentiality) guarantees expected for remote auditing. The paper further reports on a prototypical implementation and tests carried out to empirically demonstrate its operation.

Lessons learned. A straightforward combination of standard public key primitives (and distributed protocols based thereupon) can be used to construct robust logging protocols. In fact, tamper evidence and confidentiality of log data in transit and at rest can be provided by these means. A further insight regards the structure of log data. The chosen structure is, for simplicity reasons, linear; that is, the events are recorded without any organization. This causes a problem while querying log data, because in the worst case each query requires the consideration of the whole log.

Further work. Further work considers several extensions of the BBox, as well as its implementation. Firstly, we aim at employing faster algorithms to verify the integrity of log entries and, more importantly, improved methods to search for log entries. As for the search, we have been testing with (distributed) hash tables, which clearly accelerate the search but introduce a tremendous overhead for creating and managing indexes on-the-fly in settings where a large number of records are transmitted to the collector.

Overall, we identify the need for efficient data-structures in digital black boxes as the main research direction in this setting. Here, tree-structures appear to be more promising than distributed hash tables. Similar to log file audit [46], we have been experimenting with tree-structures to accelerate the retrieval of log entries. A crucial aspect is to decide on the branching criteria. Preliminary tests using object and role hierarchies show that fine-grained hierarchies, in this case for objects, lead to more efficient search trees. However, we still have to substantiate with more formal evidence.

Acknowledgments

The author cordially thanks Matthias Bernauer and Thomas Stocker for their valuable feedback and patience for extensively testing the prototypical BBox implementation, and the anonymous reviewers for their constructive reports.

References

[1] A. Carlin, F. Gallegos, IT audit: a critical business process, IEEE Computer 40 (7) (2007) 87–89.
[2] G. Müller, R. Accorsi, S. Höhn, S. Sackmann, Sichere Nutzungskontrolle für mehr Transparenz in Finanzmärkten, Informatik Spektrum 33 (1) (2010) 3–13.
[3] R. Accorsi, Business process as a service: chances for remote auditing, in: IEEE International Computer Software and Applications Conference, IEEE Computer Society, 2011, pp. 398–403.
[4] R. Teeter, M.A. an Miklos Vasarhelyi, Remote auditing: a research framework, Journal of Emerging Technology in Accounting (2010).
[5] R. Chow, P. Golle, M. Jakobsson, E. Shi, J. Staddon, R. Masuoka, J. Molina, Controlling data in the cloud: outsourcing computation without outsourcing control, in: Proceedings of the ACM Workshop on Cloud Computing Security, ACM, 2009, pp. 85–90.
[6] R. Mercuri, On auditing audit trails, Communications of the ACM 46 (1) (2003) 17–20.
[7] R. Accorsi, Safe-keeping digital evidence with secure logging protocols: state of the art and challenges, in: O. Goebel, R. Ehlert, S. Frings, D. Günther, H. Morgenstern, D. Schadt (Eds.), Proceedings the IEEE Conference on Incident Management and Forensics, IEEE Computer Society, 2009, pp. 94–110.
[8] B. Waters, D. Balfanz, G. Durfee, D. Smetters, Building an encrypted and searchable audit log, in: Proceedings of the 11th Annual Network and Distributed System Security Symposium, 2004.
[9] M. Bellare, B. Yee, Forward integrity for secure audit logs, Tech. Rep., University of California, San Diego, Dept.of Computer Science & Engineering, 1997.
[10] L. Lowis, R. Accorsi, On a classification approach for SOA vulnerabilities, in: Proceedings of the IEEE International Computer Software and Applications Conference, IEEE Computer Society, 2009, pp. 439–444.
[11] L. Lowis, R. Accorsi, Finding vulnerabilities in SOA-based business processes, IEEE Transactions on Service Computing 4 (3) (2011) 230–242.
[12] R. Accorsi, C. Wonnemann, S. Dochow, SWAT: a security workflow toolkit for reliably secure process-aware information systems, in: Conference on Availability, Reliability and Security, IEEE, 2011, pp. 692–697.
[13] R. Accorsi, C. Wonnemann, Detective information flow analysis for business processes, in: W. Abramowicz, L. Macaszek, R. Kowalczyk, A. Speck (Eds.), Business Processes, Services Computing and Intelligent Service Management, in: Lecture Notes in Informatics, vol. 147, Springer, 2009, pp. 223–224.
[14] R. Accorsi, C. Wonnemann, Strong non-leak guarantees for workflow models, in: ACM Symposium on Applied Computing, ACM, 2011, pp. 308–314.
[15] R. Accorsi, C. Wonnemann, T. Stocker, Towards forensic data flow analysis of business process logs, in: O. Goebel, R. Ehlert, S. Frings, D. Günther, H. Morgenstern, D. Schadt (Eds.), Proceedings the IEEE Conference on Incident Management and Forensics, IEEE Computer Society, 2011, pp. 94–110.
[16] R. Accorsi, T. Stocker, On the exploitation of process mining for security audits: the conformance checking case, in: ACM Symposium on Applied Computing, ACM Press, 2012, pp. 1709–1716.
[17] A. Chuvakin, G. Peterson, Logging in the age of web services, IEEE Security and Privacy 7 (3) (2009) 82–85.
[18] R. Accorsi, Automated counterexample-driven audits of authentic system records, Ph.D. Thesis, University of Freiburg, 2008.
[19] C. Lonvick, RFC3164: the BSD syslog protocol, request for comments, 2001. http://www.ietf.org/rfc/rfc3164.txt.
[20] R. Accorsi, A. Hohl, Delegating secure logging in pervasive computing systems, in: J. Clark, R. Paige, F. Pollack, P. Brooke (Eds.), Proceedings of the 3rd International Conference on Security in Pervasive Computing, in: Lecture Notes in Computer Science, vol. 3934, Springer, 2006, pp. 58–72.
[21] B. Lampson, A note on the confinement problem, Communications of the ACM 16 (10) (1973) 613–615.
[22] D. Dolev, A. Yao, On the security of public key protocols, IEEE Transactions on Information Theory 2 (29) (1983) 198–208.
[23] M. Franklin, A survey of key evolving cryptosystems, International Journal of Security and Networks 1 (1–2) (2006) 46–53.
[24] L. Lamport, Password authentication with insecure communication, Communications of the ACM 24 (11) (1981) 770–772.
[25] R. Needham, M. Schroeder, Using encryption for authentication in large networks of computers, Communications of the ACM 21 (12) (1978) 993–999.
[26] M. Abadi, R. Needham, Prudent engineering practice for cryptographic protocols, IEEE Transactions on Software Engineering 22 (1) (1996) 6–15.
[27] Automated validation of Internet security protocols and applications, 2008. http://www.avispa-project.org/.
[28] L. Lowis, A. Hohl, Enabling persistent service links, in: Proceedings of the International Conference on E-Commerce Technology, IEEE Computer Society, 2005, pp. 301–306.
[29] S. Sackmann, J. Strüker, R. Accorsi, Personalization in privacy-aware highly dynamic systems, Communications of the ACM 49 (9) (2006) 32–38.
[30] D. Yum, J. Kim, P. Lee, S. Hong, On fast verification of hash chains, in: J. Pieprzyk (Ed.), Topics in Cryptology, in: Lecture Notes in Computer Science, vol. 5985, Springer, 2010, pp. 382–396.
[31] B. Schneier, J. Kelsey, Security audit logs to support computer forensics, ACM Transactions on Information and System Security 2 (2) (1999) 159–176.
[32] D. Ma, G. Tsudik, A new approach to secure logging, ACM Transactions on Storage 5 (1) (2009) 1–21.
[33] Reliable syslog. http://security.sdsc.edu/software/sdsc-syslog/.
[34] J. Kelsey, J. Callas, Signed syslog messages, IETF Internet Draft, 2005. http://www.ietf.org/internet-drafts/draft-ietf-syslog-sign-16.txt.
[35] Syslog-ng web site. http://www.balabit.com/products/syslog_ng/.
[36] R. Accorsi, On the relationship of privacy and secure remote logging in dynamic systems, in: S. Fischer-Hübner, K. Rannenberg, L. Yngström, S. Lindskog (Eds.), Security and Privacy in Dynamic Environments, in: IFIP Conference Proceedings, vol. 201, Springer, 2006, pp. 329–339.
[37] V. Stathopoulos, P. Kotzanikolaou, E. Magkos, A framework for secure and verifiable logging in public communication networks, in: J. Lopez (Ed.), Proceedings of the Workshop on Critical Information Infrastructures Security, in: Lecture Notes in Computer Science, vol. 4347, Springer, 2006, pp. 273–284.
[38] C. Chong, Z. Peng, P. Hartel, Secure audit logging with tamper-resistant hardware, in: D. Gritzalis, S.D.C. di Vimercati, P. Samarati, S. Katsikas (Eds.), Security and Privacy in the Age of Uncertainty, in: IFIP Conference Proceedings, vol. 250, Kluwer, 2003, pp. 73–84.
[39] J. Holt, Logcrypt: forward security and public verification for secure audit logs, in: R. Buyya, T. Ma, R. Safavi-Naini, C. Steketee, W. Susilo (Eds.), Proceedings of the Australasian Symposium on Grid Computing and e-Research, in: CRIPT, vol. 54, Australian Computer Society, 2006, pp. 203–211.
[40] D. Ma, Practical forward secure sequential aggregate signatures, in: M. Abe, V.D. Gligor (Eds.), Proceedings of the ACM Symposium on Information, Computer and Communications Security, ACM, 2008, pp. 341–352.
[41] R. Curtmola, J. Garay, S. Kamara, R. Ostrovsky, Searchable symmetric encryption: improved definitions and efficient constructions, in: A. Juels, R. Wright, S.D.C. di Vimercati (Eds.), ACM Conference on Computer and Communications Security, ACM, 2006, pp. 79–88.
[42] D. Boneh, M. Franklin, Identity based encryption from the Weil pairing, SIAM Journal of Computing 32 (3) (2003) 586–615.
[43] B. Bloom, Space/time trade-offs in hash coding with allowable errors, Communications of the ACM 13 (7) (1970) 422–426.
[44] W. Xu, D. Chadwick, S. Otenko, A PKI based secure audit web server, in: IASTED Communications, Network and Information, 2005.
[45] E. Kenneally, Digital logs—proof matters, Digital Investigation 1 (2) (2004) 94–101.
[46] R. Accorsi, T. Stocker, Automated privacy audits based on pruning of log data, in: Proceedings of the EDOC International Workshop on Security and Privacy in Enterprise Computing, IEEE, 2008.