

# BBox: A Distributed Secure Log Architecture

Rafael Accorsi

Department of Telematics  
University of Freiburg, Germany  
accorsi@iig.uni-freiburg.de

**Abstract.** This paper presents BBox, a digital black box to provide for authentic archiving 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 retrieval by auditors. Besides presenting the technical underpinnings of the BBox, this paper demonstrates the authenticity guarantees it ensures and reports on the preliminary deployment figures.

**Keywords.** Distributed log architecture, public key cryptography.

## 1 Introduction

The growing number of national and international compliance requirements emphasizes the importance archiving of business processes transactions, where records are analyzed as part of an audit to corroborate or refute potential violations of compliance rules [7]. To this end, authentic records are essential to guarantee reliable accountability and non-repudiation of actions [19].

While the demand for digital black boxes as a means to guarantee the authenticity of records is evident [21], 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 [20]. 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 [5].

This paper presents BBox, a digital black box to provide for authentic system records in distributed systems. Authenticity means on the one hand the *integrity* of records and, on the other, their *confidentiality*. While integrity is an indisputable requirement, the importance of confidentiality stems largely from the context within which the BBox is employed. Generally, it is simply unreasonable to store entries in clear, even though this complicates the log files search [28]. For these requirements, the BBox addresses the collection, the transmission, the storage and the retrieval of records. Among others, the BBox guarantees:

- *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 [6].
- *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.

The BBox builds upon public key cryptography and trusted computing modules to ensure the authenticity of log records in distributed systems. It is being deployed as a component of a business process management system (BPMS) to guarantee authentic archiving during the workflow execution in a service-oriented architecture. 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 [9]. The following presents the high-level cryptographic building blocks, not their particular implementation details in the BPMS.

This paper is organized as follows. §2 builds the core of the paper, presenting the architecture, components and operation of the BBox. §3 addresses the retrieval of log views. §4 reports on the security analysis of the BBox. §5 concludes the paper and discusses further research topics for digital black boxes.

## 1.1 Terminology and Related Work

A log architecture 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 portions of the log file. 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*.

Existing secure log services can be categorized in those adding security functionalities to the syslog [26] and those based on the Schneier-Kelsey scheme [24]. 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 [22]. The syslog-sign extends syslog with signature blocks preventing tampering of log data in transit [13]. 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 clear. Distributions of \*nix systems have been equipped with

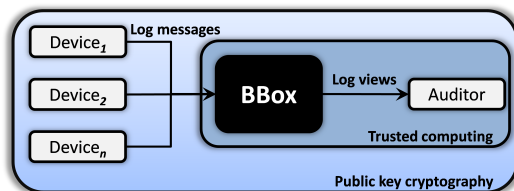


Fig. 1. The high-level architecture of the BBox logging.

syslog-ng, the successor of the syslog [27]. Besides reliable transfer over TCP, it also supports IPv6 and encrypted log message transfer using the TLS protocol.

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 [1]. Stathopoulos et al. present the application of Schneier-Kelsey’s scheme for the telecommunication setting [25] and Chong et al. employ it for DRM [8]. These proposals exhibit a similar vulnerability: the “tail-cut” attack differently reported in [5] and [12]. Logging services based on the Schneier-Kelsey scheme employ hash chains [15] 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.

Xu et al. propose the SAWS architecture, the secure audit web server [29]. 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 [18]. 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 clearly succumbs 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 [14].

## 2 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

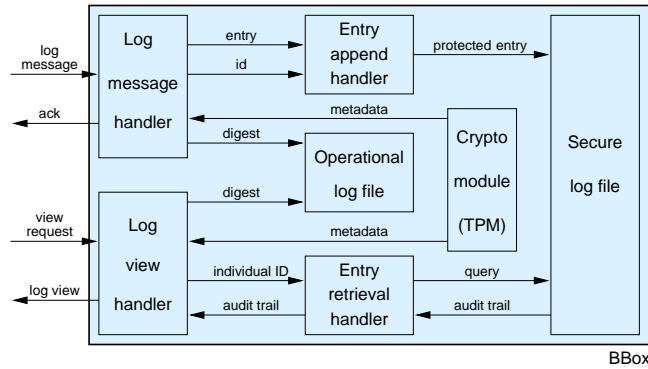


Fig. 2. BBox: Architecture, components and main information flows.

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 auditors may query the BBox to obtain *log views*, i.e. audit trails containing the all log entries matching the search criteria.<sup>1</sup> 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.

## 2.1 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 §2.3.
- *Secure log file*. This is the container where events are securely recorded after being prepared by the EAH.

<sup>1</sup> 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.

- *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 metadata 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-)initialisations and shutdowns.

**Notation.** The presentation of the BBox employs the following notation:  $d_i$  denotes the  $i$ th device;  $P_i$  refers to the *payload* of the  $i$ th 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  $i$ th *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  $K_s^{-1}$ .

## 2.2 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  $K_{\text{BBox}}$  and  $K_{\text{BBox}}^{-1}$  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  $G_0 = Hash(K_{\text{BBox}}^{-1})$  which is used as basis for the secure logging service (see §2.3).

The third step at initialising 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. 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.

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

**Table 1.** Excerpt of a BBox' DAKL table.

The fourth and final step consists in opening the 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

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

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  $K_{\text{BBox}}^{-1}$  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 timestamp 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.

### 2.3 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 §2.2) 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 [11]. 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

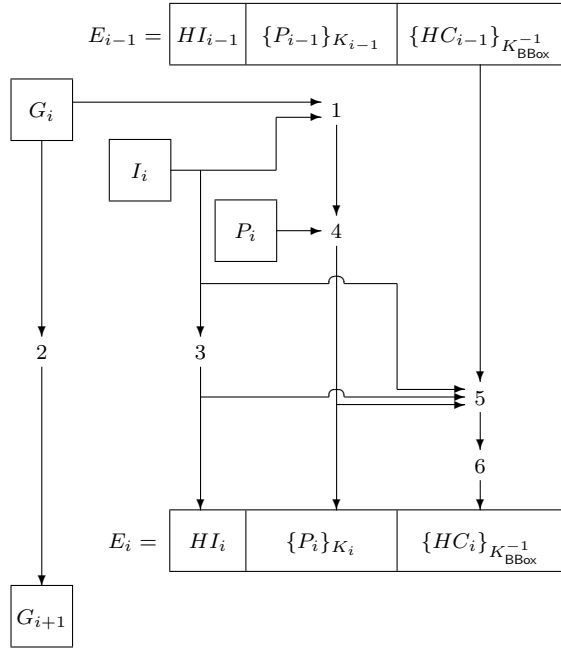


Fig. 3. Appending an entry to the secure log file.

entry. Hence, the keys  $K_i, K_{i+1}, \dots$  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 [15]. 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 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  $\{HC_i\}_{K_{BBox}^{-1}}$ ).

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.  $HC_i = Hash(I_i, HI_i, \{P_i\}_{K_i}, \{HC_{i-1}\}_{K_{\text{BBox}}^{-1}})$  is the  $i$ th link of the hash chain.
6.  $\{HC_i\}_{K_{\text{BBox}}^{-1}}$  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  $E_i = (HI_i, \{P_i\}_{K_i}, \{HC_i\}_{K_{\text{BBox}}^{-1}})$ , 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.

## 2.4 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 Alg. 1, which roughly speaking “traverses” the hash-chain seeking for broken links.

Alg. 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  $G_{\text{current}}$ . 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 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  $G_{\text{current}}$ ).

Alg. 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  $i$ th

---

**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  $K_{\text{BBox}}^{-1}$ )
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

```

---

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 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).

---

**Algorithm 2** Entry Selection

---

**Require:**  $I$ ,  $LogFile$ ,  $OpLogFile$ ,  $G_0$ ,  $G_{current}$ **Provide:**  $T$ , Authentication\_Failure

```

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

```

---

### 3 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 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 correct the BBox. Thus, *mutual* authentication between requesting individuals and the BBox is necessary.
- *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.

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 timestamp) and sending the resultant tuple  $T, M$  to the requesting individual.

The following focuses on the selection of entries. Details about the mutual authentication and remote attestation are not given here. Mutual authentication is prototypically achieved with a variant of the Needham-Schroeder protocol and attestation employs the standard primitives.

**Selection of log entries.** Alg. 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 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  $A$ .

## 4 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.

### 4.1 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 §2.2) has been verified in the AVISPA tool for authentication and replay-freeness properties. 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 [10], no such attack could be found. Moreover, since the messages are encrypted, confidentiality against an eavesdropper is preserved.

The use of timestamps 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 timestamps 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 [2].

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 [17].

## 4.2 Log Data at Rest

Two security properties are relevant for log data at rest: confidentiality and integrity. The former is achieved by encrypting the payload of the entries with the symmetric key  $K$ . This key is generated using the evolving key  $G$  (derived from 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  $i$ th 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.

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 [23], 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 case of an attack, the events of the customers could be linked and their identity could be disclosed.

With regard to integrity guarantees, Alg. 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, move 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 Alg. 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.** *Alg. 1 is correct iff it does not exhibit false negatives.*  $\dashv$

**Theorem 1.** *Alg. 1 is correct with regard to authentication.*  $\dashv$

To show Theorem 1, every attack tampering attempt on an initially authentic log file must be examined, thereby demonstrating that they are detected. Due to space constraints, below we only demonstrate one case of the proof – for modification attacks. Other attacks are shown in a similar manner [3].

*Proof (for modification attacks).* Let *LogFile* be an authentic log file consisting of a sequence of entries  $E_0, \dots, E_n$  constructed according to §2.3, where each entry  $E_j$ , with  $0 \leq j \leq n$ , assumes the form  $E_j = (HI_j, \{P_j\}_{K_j}, \{HC_j\}_{K_{\text{BBox}}^{-1}})$ .

*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).

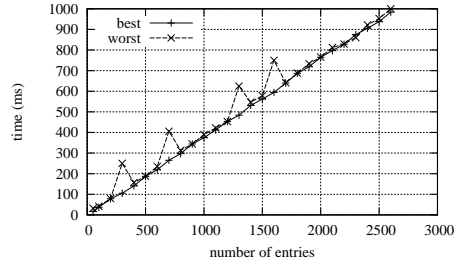
(1.2) Overwriting the encrypted payload of  $E_j$ : irrespective of whether the attacker reuses the payload field of an existing entry  $E_k$  or generates a new payload field from the items he possesses, such an overwriting leads to a broken hash chain, as the expected value for link of the hash chain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

(1.3) Overwriting the hash chain link of  $E_j$ : the attacker either overwrites the link  $\{HC_j\}_{K_{\text{BBox}}^{-1}}$  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  $K_{\text{BBox}}^{-1}$ , he cannot generate a legitimate link that passes the signature check (Line 9).  $\square$

The following provides an intuition of how the other violations of integrity properties are detected by the algorithm. *Inclusions* are detected using the signature appended to the link of the hash chain: since the attacker does not possess  $K_{\text{BBox}}^{-1}$  and since he cannot cause collision of hash links, the attacker cannot produce valid signatures. A broken hash chain or a chain shorter than expected indicates that the some link or part of the log file has been *removed*.

### 4.3 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



**Fig. 4.** Time necessary to authenticate a secure log file.

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 [16] – to identify potential vulnerabilities. Moreover, we conducted man-in-the-middle and tampering attacks to observe how the BBox behaves.

Among others, 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.

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 the algorithm. The BBox succeeded in authenticating the log files and detecting the attack attempts. For illustration, Fig. 4 depicts the runtime necessary to authenticate sample log files. (These runtime figures were obtained in a standard desktop PC with 1.92GHz, 512MB of RAM and 250GB hard disk operating under MS-Windows and Java version 1.6; each entry payload amounts to 50Kb.) 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 raising number of log messages (between 500 and 2500 messages per minute; step 500). Legitimate entries sent after 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 to a bottleneck in the BBox architecture.

## 5 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 reliable accountability. It is shown that tamper evidence and confidentiality of log data in transit and at rest can be provided and also the limits in which this is possible. Besides the implementation of its components with a business process management system, the algorithms behind the BBox are being further developed to improve performance. In particular, 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 latter, we have been testing with (distributed) hash tables, which clearly accelerate the search but introduce a tremendous overhead for creating 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 [4], 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.

## References

1. R. Accorsi. On the relationship of privacy and secure remote logging in dynamic systems. In S. Fischer-Hübner et al. (eds.) *IFIP Conf. Proceedings* vol. 201, pp. 329–339. Springer, 2006.
2. R. Accorsi and A. Hohl. Delegating secure logging in pervasive computing. In J. Clark et al. (eds.) *Security in Pervasive Computing, LNCS* vol. 3934, pp. 58–72. Springer, 2006.
3. R. Accorsi. Automated counterexample-driven audits of authentic system records. Ph.D. dissertation, University of Freiburg, 2008.
4. R. Accorsi and T. Stocker. Automated privacy audits based on pruning of log data. *IEEE Enterprise Distributed Object Computing Conference*, pp. 175–182, 2008.
5. R. Accorsi. Safe-keeping digital evidence with secure logging protocols: State of the art and challenges. In O. Goebel et al. (eds.) *Incident Management and Forensics*, pp. 94–110. IEEE, 2009.
6. M. Bellare and B. Yee. Forward integrity for secure audit logs. Tech. report, U of California, San Diego, Dept. of Computer Science & Engineering, 1997.
7. A. Carlin and F. Gallegos. IT audit: A critical business process. *IEEE Computer*, 40(7):87–89, 2007.

8. C. Chong, Z. Peng, and P. Hartel. Secure audit logging with tamper-resistant hardware. In D. Gritzalis et al. (eds.) *IFIP Conf. Proceedings* vol. 250, pp. 73–84. Kluwer, 2003.
9. A. Chuvakin and G. Peterson. Logging in the age of web services. *IEEE Security and Privacy*, 7(3):82–85, 2009.
10. D. Dolev and A. Yao. On the security of public key protocols. *IEEE Transactions on Information Theory*, 2(29):198–208, 1983.
11. M. Franklin. A survey of key evolving cryptosystems. *International Journal of Security and Networks*, 1(1-2):46–53, 2006.
12. J. Holt. Logcrypt: Forward security and public verification for secure audit logs. In R. Buyya et al. (eds.) *Australasian Symposium on Grid Computing and e-Research, CRIPT* vol. 54, pp. 203–211, 2006.
13. J. Kelsey and J. Callas. Signed syslog messages. IETF Internet Draft, 2005.
14. E. Kenneally. Digital logs - Proof matters. *Digital Investigation*, 1(2):94–101, 2004.
15. L. Lamport. Password authentication with insecure communication. *Commun. ACM*, 24(11):770–772, 1981.
16. L. Lowis and R. Accorsi. Finding vulnerabilities in SOA-based business processes. *IEEE Transactions on Services Computing*, 2010. (To appear)
17. L. Lowis and A. Hohl. Enabling Persistent Service Links. *IEEE Conference on E-Commerce Technology*, pp. 301–306, 2005.
18. D. Ma and G. Tsudik. A new approach to secure logging. *ACM Transactions on Storage*, 5(1):1–21, 2009.
19. G. Müller, R. Accorsi, S. Höhn, and S. Sackmann. Sichere Nutzungskontrolle für mehr Transparenz in Finanzmärkten. *Informatik Spektrum*, 33(1):3–13, 2010.
20. R. Mercuri. On auditing audit trails. *Commun. ACM*, 46(1):17–20, 2003.
21. R. Oppliger and R. Ritz. Digital evidence: Dream and reality. *IEEE Security and Privacy*, 1(5):44–48, 2003.
22. Reliable syslog. <http://security.sdsc.edu/software/sdsc-syslog/>.
23. S. Sackmann, J. Strüker, and R. Accorsi. Personalization in privacy-aware highly dynamic systems. *Commun. ACM*, 49(9):32–38, 2006.
24. B. Schneier and J. Kelsey. Security audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159–176, 1999.
25. V. Stathopoulos, P. Kotzanikolaou, and E. Magkos. A framework for secure and verifiable logging in public communication networks. In J. Lopez (ed.) *Critical Information Infrastructures Security, LNCS* vol. 4347, pp. 273–284. Springer, 2006.
26. Syslog. <http://www.syslog.org/>.
27. Syslog-ng web site. [http://www.balabit.com/products/syslog\\_ng/](http://www.balabit.com/products/syslog_ng/).
28. B. Waters, D. Balfanz, G. Durfee, and D. Smetters. Building an encrypted and searchable audit log. In *Network and Distributed System Security*, 2004.
29. W. Xu, D. Chadwick, and S. Otenko. A PKI Based Secure Audit Web Server. In *IASTED Communications, Network and Information*, 2005.
30. D. Yum, J. Kim, P. Lee and S. Hong. On fast verification of hash chains. In J. Pieprzyk (ed.) *Topics in Cryptology, LNCS* vol. 5985. Springer, 2010