Log Data as Digital Evidence: What Secure Logging Protocols Have to Offer?

Rafael Accorsi
Dept. of Telematics, University of Freiburg, Germany
raccorsi@acm.org

***Abstract*—While log data are being increasingly used as digital evidence in judicial disputes, the extent to which existing secure logging protocols used to collect log data fulfill the legal requirements for admissible evidence remain largely unclear. We elucidate the necessary secure requirements for digital evidence and extensively survey the state of the art secure logging protocols, thereby demonstrating that none of the current proposals fulfills the necessary conditions for admissible evidence.**

I. Introduction

Log data record the "dynamics" of a system, i.e. the *events* that change a system's state, including information about when an event occurs, which subject triggers it, and which objects it involves. Rather than a static snapshot of a system's state, log data allow the reconstruction of complex event chains, thereby being a central source of digital evidence for legal disputes.

To be admitted as evidence, log data must fulfill several legal requirements regarding the *architecture* and the *cryptographic operations* used to collect and protect log data. While the former are well-studied [17], for the latter unclarity prevails. Two questions are of foremost relevance here: (1) how legal requirements translate into security requirements common to system engineers and (2) which and how security requirements are satisfied by the existing secure logging protocols prescribing such cryptographic operations.

We address these questions. First, in Section II we elucidate what security requirements are necessary – yet not sufficient – for logging protocols to render log data as admissible digital evidence, expressing them as usual protection goals of computer security. Second, based on a threat model for logging protocols introduced in Section III and on a comprehensive survey of state of the art secure logging protocols given in Section IV, in Section V we answer the question of what secure logging protocols have to offer for admissible evidence.

This is the first thorough analysis of existing secure logging protocols for digital evidence. Focusing on their operation, it reveals that state of the art protocols fail to lay a basis for admissible digital evidence. We observe two issues: first, existing proposals focus either one the transmission or on the storage of log data, whereas both phases must be equally covered. Second, even protocols designed for the respective phases contain vulnerabilities that lead to erroneous evidence, so that log data thought as being authentic are in fact manipulated.

Logging architecture and log files: We assume the architecture proposed in [19] based on principals taking the roles:

- *Device:* senses the events and generates the corresponding log messages. In some protocols, the device applies cryptographic operations to protect log messages, e.g. by signing them for entity authentication.
- *Relay (optional):* receives log messages and forwards them to a relay or collector. Relays modify the headers of the corresponding datagrams, not messages' contents.
- *Collector:* receives log messages and generates the corresponding log file (also called "audit trail"). The collector may apply cryptographic operations to protect recorded entries against, e.g., illegal reading and deletion.

Two other 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 replaced by the new ones. Below, we assume that log files are sequential.

II. From Legal to Security Requirements

If evidence gained from log data is to influence the outcome of a legal dispute, it must be *admissible* [14]. Admissibility is a legal concept that prescribes requirements to judge the acceptance of any kind of evidence – be it digital or not. Kenneally surveys several ways in which log data can be eventually admitted as evidence [16]. Underlying these postures, the basic evidentiary tenet governing admissibility determinations is that there are guarantees of *trustworthiness* associated with the evidence, otherwise it can be challenged [10].

To-date there is no consensus on how to characterize trustworthiness as *security* requirements. Clearly, collected log data must be a faithful representation of the events communicated by the devices. Moreover, once stored, it should not be possible to undetectably alter log data. But are these the sole requirements to be guaranteed by secure logging protocols? If not, what are the others?

We translate Cohen's "trustworthiness" requirement into security requirements that guarantee the *authenticity* of audit trails, thereby distinguishing between the requirements on the transmission of log messages and those on their storage.

A. Transmission Phase

In this phase, log *messages* are in transit between the device and the collector, where the following requirements are set:

- *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 in cases where the collector communicates with $\mathcal{V}$ and $\mathcal{T}$.

B. Storage Phase

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.

These security requirements are necessary, yet not sufficient. Complementary, non-architectural measures protecting, e.g., the access to log files and the isolation of the logging process to avoid cover channels are necessary as well. However, they traditionally fall outside the scope of secure logging protocols.

III. Attacker Models for Logging Protocols

An *attacker model*, i.e. a model of the capabilities of attackers, is necessary when analyzing the threats to the audit trails and the protection offered by secure logging protocols.

One of the shortcomings in analyzing digital evidence and their admissibility is the absence of well-accepted attacker models for audit trails. Nevertheless, [2], [21], and [27] give informal descriptions of attacker capabilities. Following the distinction for security requirements above, we separate the attacker's capabilities for the transmission and storage phases. Analyses of logging protocols must must consider both phases.

A. Attacker Model for the Transmission Phase

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 [11]. 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 submessages. The attacker is also computationally bounded, being subject to the same complexity constraints for cryptographic computation as legitimate principals.

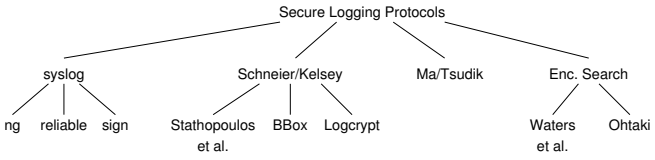

Fig. 1. Classes of secure logging protocols.

B. Attacker Model for the Storage Phase

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 the 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.

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.

IV. Secure Logging Protocols: State of the Art

We group the existing logging protocols in four classes according to the techniques they build upon, as in Fig. 1. The "Syslog" class consists of protocols based on the syslog; the "Schneier/Kelsey" class consists of protocols extending that of [27]; the "Ma/Tsudik" class includes the novel logging techniques in [21]; the "Encrypted Search" class contains protocols to search encrypted log data.

A. Syslog Class

The syslog [19] standard in several *nix distributions, was *not* developed to provide for secure logging. (It transmits and stores messages in clear-text and uses the unreliable UDP to deliver log messages.) This led to a number of extensions of the traditional syslog focusing on the transport of data from the device to the collector, thus neglecting the storage phase.

1) syslog-ng: syslog's new generation [29] provides a secure, reliable log service, being backward compatible with syslog. The improvements against the syslog are: use of TCP as transport protocol; support to IPv6 protocol; and encrypted transmission with the TLS protocol.

2) syslog-sign: It adds origin authentication, message integrity, replay resistance, and detection of missing messages to syslog [15]. For this, it uses cryptographically signed messages that contain the signatures of previously sent syslog messages. Fig. 2 depicts how such "signature blocks" are created. (The necessary cryptographic keys are generated in the initialization

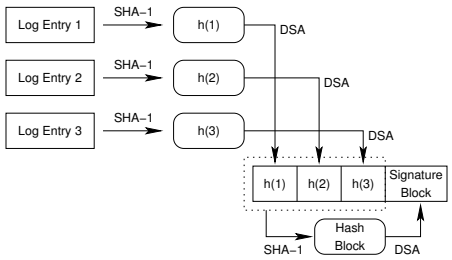

Fig. 2. Creation of signature blocks in syslog-sign.

phase.) The syslog-sign does not provide confidentiality during the transmission of log data, nor does it account for encrypted storage of log entries. Since signature blocks are supposed to be deleted after the authentication, tamper evidence is only partially fulfilled. Implementations of syslog-sign do not exist.

3) reliable syslog: The reliable syslog is based on the RFC 3195 [23] (as opposed to RFC 3164) and aims to implement reliable delivery for syslog messages. It uses TCP as transport protocol and provides for device authentication, mechanisms to protect the integrity of log messages, and protection against replay attacks. The San Diego Supercomputer Center implemented the reliable syslog [26].

B. Schneier-Kelsey Class

Building on the work of Bellare and Yee [5], Schneier and Kelsey [27] devise a logging protocol for the storage phase. They employ *hash chains* [18] and *evolving cryptographic systems* [12] as cryptographic techniques to protect of audit trails. A hash chain is a successive application of a cryptographic hash function to a string. Evolving keys is an encryption technique in which (usually symmetric) keys evolve over time, possibly as a function of the previous keys. The timespan in which the key is kept the same is called an "epoch." The idea is to limit the impact of key compromise: attackers who obtain a key can only decrypt entries in the corresponding epoch.

Based on these primitives, the authors provide algorithms to create/close an audit trail, append entries to it, and authenticate its entries. (Authentication checks audit trails for tampering.)

Fig. 3 depicts an entry in Schneier and Kelsey. The field W_i is an authorization mask that controls the access to the contents of the entry L_i. (Only subjects authorized in W_i gain access to the contents E_i of L_i.) $\{E_i\}_{K_i}$ stands for the ith event E_i encrypted with the unique, symmetric *evolving* key K_i.

The key K_i is generated by hashing the authorization mask W_i and the entry authentication key A_i. After generating K_i and computing the value Z_i (see below), a new entry authentication key A_{i+1} for the next entry is generated, which is then used to generate K_{i+1} and provide for key evolution. Thus, each entry corresponds to an epoch. Besides assuming the entry authentication keys A are kept secret, the authors assume that A_i and K_i are irretrievably deleted after use.

The field Y_i is the ith link of the hash chain. To intertwine entries of the audit trail, the hash chain link Y_i is generated by hashing the fields W_i, $\{E_i\}_{K_i}$, and the link Y_{i-1} of the

$$L_i := \boxed{W_i} \boxed{\{E_i\}_{K_i}} \boxed{Y_i} \boxed{Z_i}$$

Fig. 3. Format of a log entry in Schneier and Kelsey [27].

previous entry L_{i-1}. In doing so, a verifier can assert the absence of tampering by checking the chain for *broken links*. Finally, the field Z_i stands for the message authentication code (MAC) – computed on the basis of A_i – to authenticate the entry. (Z_i is not included in the generation of the link Y_i.)

While Schneier and Kelsey provide no implementation of their protocol, Chong et al. embed the protocol in a tamper resistant hardware [8] and Accorsi and Hohl focus on the delegation of logging tasks using trusted computing [3].

Informal threat analyses of the Schneier and Kelsey protocol revealed attacks in which modifications of the audit trail could go undetected, thereby violating entry integrity and tamper evidence. This led to the following extensions of [27]:

1) Stathopoulos et al.: In [28], the authors apply the logging protocol in the public communication networks. Their extension aims to prevent internal attacks, in which a fraudster could reconstruct parts of the audit trail in way that tamper could not be detected. (This attack is possible if the attacker knows some A_j.) They introduce a "regulatory authority" (*RA*) to assure that the collector follows the protocol accordingly.

To this end, the authors employ an approach similar to the signature blocks used in the syslog-sign (see Section IV-A2): periodically, *RA* receives a signature block from the collector, storing these blocks for further analysis. If suspicious arises, the signature blocks is recomputed and compared with those stored at the *RA*. The authors also informally analyze the security guarantees provided by their logging protocol, demonstrating that the protocol is robust against such attacks.

*2) "*BBox*":* For Alles et al. [4] and Oppliger and Ritz [25], trustworthy digital evidence requires a kind of "black box" logging, similar to flight recorders used in airplanes. Using this picture, Accorsi devises the BBox: a digital black box [2, Chap. 3]. (A previous version of his work appears in [1].)

By means of an informal threat analysis, the author demonstrates that truncation attacks are possible. The attacker may remove the last n entries of the audit trail in a way that goes undetected at the authentication. (Truncation goes undetected because it does not lead to a broken hash chain.) To fix this vulnerability, as well as the one introduced by Stathopoulus, Accorsi modifies the protocol of Schneier and Kelsey and applies trusted computing modules and novel cryptographic approaches to digital evidence [22]. In doing so, he focuses not only on the storage and authenticated, keyword-based retrieval of audit trails, but also defines security protocols for the transmission of log data between the devices, the BBox, and external auditors.

Assuming a PKI, each log message is encrypted and signed for the transmission to the collector. The BBox ensures the origin and integrity of the log message, while the replay attacks are addressed by means of sequence numbers and timestamps.

As for the storage, the author removes the field Z_i of an entry, requiring the hash chain links to be signed by the BBox. The authentication of the audit trail detects truncation and other kinds of tampering.

Accorsi (to a certain extent formally) demonstrates the security properties provided by the BBox and successfully carries out a series of tests with the resultant implementation.

3) Logcrypt: Holt devises Logcrypt, a logging protocol that provides strong cryptographic assurances that data stored by a collector cannot "be modified after the compromise without detection" [13]. The most significant improvement is the ability to use a public key cryptography in computing the field Z. Using the symmetric techniques proposed by [27], any entity $\mathcal{V}$ who wishes to verify the a log must possess the secret A used as parameter of the MAC function. This secret also gives the entity the ability to falsify the log entries, which would render evidence gathered this way utterly wrong. Public key – specifically, Identity-Based Encryption, see Section IV-D – allows signatures to be created with one key and verified with another. Thus, the major aspect of Logcrypt is the replacement of MAC with signatures, in a way similar to Accorsi [2].

Other improvements described in [13] include a method of securing multiple, concurrently maintained log files using a single initial value, and a method of aggregating multiple log entries to reduce latency and computational load. Holt provides algorithms to this end and reports on performance analysis obtained with a prototypical implementation.

C. Ma and Tsudik Class

Ma and Tsudik [21] propose a logging protocol for the storage phase that employs a novel authentication technique called "Forward-Secure Sequential Aggregate" (FssAgg) [20].

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.

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 [27], *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.

At a given timepoint an authenticated log file consists of two parts: audit trail $[L_1, \ldots, L_{i-1}]$ and two FssAgg authentication MACs $\mu_{\mathcal{T},i-1}$ and $\mu_{\mathcal{V},i-1}$. In addition to that, the collector possesses two random symmetric keys A_i and B_i, which are created during the initialization phase and updated – by means of a cryptographic hash function – for each new entry. The initial keys A_1 and B_1 are communicated during the initialization phase to a semi-trusted verifier $\mathcal{V}$ and the trusted verifier $\mathcal{T}$, respectively.

When the ith event occurs, the collector $\mathcal{U}$ creates and appends the entry L_i to the audit trail, updating the authentication tags $\mu_{\mathcal{V}}$ and $\mu_{\mathcal{T}}$ as follows:

1) $\mathcal{U}$ generates a MAC for the verifier $\mathcal{V}$ $mac_{A_i}(L_i)$. It then computes $\mu_{\mathcal{V},i} = \mathcal{H}(\mu_{\mathcal{V},i-1}||mac_{A_i}(L_i))$, where $\mathcal{H}$ is an one-way, collision-resistant hash function acting as an aggregator.
2) $\mathcal{U}$ updates the second FssAgg MAC (for $\mathcal{T}$) in the same way using the value B_i: $\mu_{\mathcal{T},i} = \mathcal{H}(\mu_{\mathcal{T},i-1}||mac_{B_i}(L_i))$.
3) $\mathcal{U}$ evolves both keys A_i and B_i, applying a hash function to obtain the values A_{i+1} and B_{i+1}. $\mathcal{U}$ irretrievably deletes the prior keys.

The authentication tag $\mu_{\mathcal{V}}$ can be "unrolled" as:

$$\mu_{\mathcal{V},i} = \mathcal{H}(\mathcal{H} \ldots \mathcal{H}(\mu_{\mathcal{V},i}||mac_{A_1}(L_1)) \ldots ||mac_{A_i}(L_i)),$$

where the same holds true for $\mu_{\mathcal{T},i}$ with the symmetric key B_i. That is, the final aggregator signatures are a function of (all!) the previous aggregators, thereby ensuring unforgeability.

Besides presenting the design of the logging protocol, the authors also report on its implementation, analyzing the performance of the implementation in different settings.

D. Encrypted Search Class

This class of logging protocols is concerned with the secure storage of encrypted audit trails and efficient retrieval of entries from these trails. We classify these 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 [7], and those approaches based on *Bloom Filters*, i.e. probabilistic data structures to test whether an element is a member of a set [6].[1]

1) Waters et al.: They devise protocols for the search of encrypted audit trails using IBE [30]. In IBE, public keys can be arbitrary strings; private keys are derived from public keys through a system-wide master secret known (only!) to a trusted authority. In Waters et al., a designated trusted party $\mathcal{T}$ – the so-called *key escrow agent* – constructs keyword search capabilities which allows (less trusted) parties $\mathcal{V}$ – the so-called *investigators* – to search for and decrypt entries matching a given keyword. The escrow agent can either carry out the retrieval himself or delegate this task to the investigators.

The server generating the audit trail encrypts the entries with the public keys corresponding to the keywords that are derived from those entries. The key escrow agent holding the IBE master secret can construct a search capability for a given keyword as the private key corresponding to the given keyword, whereas additional security properties of the Boneh and Franklin's scheme hold.[2]

The audit trail consists of a set of entries $\{R_0, \ldots, R_n\}$ built up as depicted in Fig 4: $E_{K_i}(m_i)$ stands for the encryption of the ith event m under a key K_i. $H(R_{i-1})$ is the hash of the previous entry, thereby generating a hash chain. $c_{w_a}, c_{w_b}, c_{w_c}$ represent the information about the keywords w_a, w_b, w_c that index the contents m_i of the entry R_i.

The relation between the keywords $w_a, \ldots, w_c$ and the encryption key K_i is of foremost importance for entry retrieval. To construct a searchable audit trail, the key escrow server first extracts keywords of the entry. Next, it encrypts the entry m_i

[1] While false positives are possible in Bloom Filters, false negatives are not.
[2] Attackers cannot tell which public key was used to create a cypher-text.

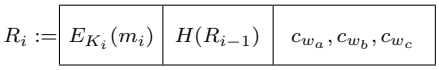

Fig. 4. Format of a log entry in Waters et al. [30].

using the key K_i derived from the keywords, so that for each keyword w, the server computes the IBE c_i of the string $(\mathbf{flag}|K)$ using the w as the public key the IBE setup P known by the server, where **flag** is a constant bit-string with length l.

Retrieval consists of the search and the decryption of entries. Assuming a search for the keyword W, the escrow agent constructs the corresponding capability d_w as the IBE private key for the string w. For each log entry and for each c the investigator attempts to IBE-decrypt c using the private key d_w. If the prefix of the result matches **flag**, then the investigator extracts K as the remainder of the result. If none of the results contain **flag**, the log entry does not match, moving to the next entry. Whenever one of the results match, the capability holder computes K to decrypt $E_K(m)$. Here, the cryptographic properties of the underlying IBE scheme prevent some investigator with capability d_w from deriving a capability $d_{w'}$ to access entries containing the keyword w'.

This protocol has been implemented and analyzed in [30].

2) Ohtaki: Similar to Waters et al. [30], Ohtaki proposes an approach based on Bloom Filters for the partial disclosure of audit trails [24]. However, in contrast to [30], Ohtaki allows for Boolean searches consisting of AND and OR operators.

The use of Bloom filters is for efficiency: they reduce the storage space for the encoded index. A Bloom filter is a probabilistic data structure to test whether an element is a member of a set [6]. Filters are represented by an array of m bits. There are also r independent hash functions $h_1, \ldots, h_r$ that take all inputs and produce outputs that are distributed uniformly over a range $[1 \ldots m]$. An arbitrary set $S = \{e_1, \ldots, e_n\}$ is represented with the m-bit array as follows:

Filter construction: Initially, every array position is set to 0. To add an element e, independent hash functions of the element are calculated and array bits at position $h_1(e), \ldots, h_r(e)$ are set to 1. This procedure is repeated to each $e_n \in S$.

Deciding set membership: To decide whether $x \in S$, the bits at positions $h_1(x), \ldots, h_r(x)$ are checked. If *any* selected bit is 0, x is *not* a member of S; if all the checked bits are 1, then x is a member of the set S. There is some probability of false positives: x appears to be in S but is actually not a member of it. This happens when all bits corresponding to x were set by other elements in S. The false positive rate can be regulated through careful selection of h, r and m.

The architecture proposed by Ohtaki foresees principals taking one of the following roles: the *administrator* manages the access to the log file and the *searcher* (or *auditor*) retrieves log entries for a keyword. Assuming a PKI, the administrator generates a pair of keys K_P and K_S, publishes the public key K_P, and sets the parameters h, r and m.

For every incoming log entry L^i with unique record identifier ID^i, the administrator generates two kinds of data: first the encrypted data V^i for partial disclosure; second, the Bloom filter BF^i used for the search and retrieval of entries. In detail, the encryption key K^i used to encrypt L^i is different for each entry, so that entry-wise decryption is possible. To this end, the symmetric key K^i is derived from the record identifier ID^i and the administrator's private key K^i, for each entry L.[3]

For the Bloom filter generation, the data used for matching is built by inserting *all* possible combinations of keywords and Boolean operators AND and OR, whereas a normalization step removes equivalent index patterns. Each normalized pattern w is converted into a "coded word" T_w, which is inserted to the Bloom Filter BF^i after the concatenation with the corresponding record identifier ID^i, denoted $ID^i||T_w$.

To obtain the contents of entries, the auditor receives the whole encrypted audit trail and communicates the keywords of interest to the administrator. Let W denote the normalized keyword pattern; the administrator converts W into the a coded keyword T_W, using to this end his private key K_S. With the encoded keyword T_W and the encrypted audit trail, the auditor queries the audit trail for matching entries. Here, the Bloom filter of each entry is verified against the concatenation of record identifier and coded keyword. That is, the auditor computes r hash functions of $ID^i||T_W$. If any of BF^i is 0, the entry does not match the search criterion. If all the bits are 1, the entry matches the keyword and, for the set of entries fulfilling the search criterion, the administrator must generate the keys allowing the auditor to decrypt them.

Ohtaki analyzes the probability of false positives and its relationship to the parameters of the Bloom filter, demonstrating that the probability of false positives already in simple Bloom filters amounts to less than 0,01%

V. What Logging Protocols Offer

Table I relates the security requirements for digital evidence of Section II and the logging protocols presented in Section IV. Each of the protocols is deployed for a particular trust model. The extensions of the Syslog, for example, assume a trusted collector, whereas the transportation medium is malicious. Hence, digital evidence generated by, e.g. syslog-ng, can be challenged on the basis that is does not protect stored log data. From this perspective, except for the BBox, all other protocols fail to address the whole spectrum of security requirements.

Problematic is the fact that even focusing on the strengths of each class, one finds vulnerabilities that lead to the violation of security requirements. The truncation and the replacement attacks upon the Schneier and Kelsey [27] protocol exemplify how subtle the design of logging protocols can be. Similarly, the BBox architecture [2] – omitted due to space constraints – builds on a number of assumptions that rely on strong tamper resistance on the side of the devices.

Secure logging protocols in the Encrypted Search class do not address the information flows or entropy that might happen when an external auditor is allowed to search through

[3] Precisely, for some entry L^i, the corresponding key is obtained as follows: $K^i = K_S(ID^i)$, where K_S stands for the public key encryption with K_S. The cypher-text V^i is generated by encrypting L^i with K^i.

TABLE I
SECURE LOGGING PROTOCOLS AND THE SECURITY REQUIREMENTS THEY FULFILL.

Secure logging protocol	**Security Requirements**							
	Transmission phase					**Storage phase**		
	confidentiality	or. authentication	integrity	uniqueness	rel. delivery	accountability	integrity	confidentiality
syslog	no	no	no	no	no	no	no	no
syslog-ng	yes	no	yes	no	yes	no	no	no
syslog-sign	no	yes	yes	yes	no	no	no	no
reliable syslog	yes	yes	yes	yes	yes	no	no	no
Schneier/Kelsey	no	no	no	no	no	yes	no	yes
Stathopoulus et al.	no	no	no	no	no	no	no	yes
BBox	yes	yes	yes	yes	yes	yes	yes	yes
Logcrypt	no	no	no	no	no	yes	yes	yes
Waters et al.	no	no	no	no	no	no	yes	yes
Ohtaki	no	no	no	no	no	yes	yes	yes
Ma/Tsudik	no	no	no	no	no	yes	yes	yes

the whole encrypted file for particular keywords. Moreover, in possession of the cypher-text, an attacker can infer other keywords and conduct chosen cypher-text attacks, thereby compromising the confidentiality of other entries.

Finally, the Ma/Tsudik class is based aggregated signatures for which correctness proofs are still missing. Experience with hash chains in [27] and more generally in the development distributed protocols (see [9]) shows that such subtle mistakes are rule rather than the exception.

Taking stock, existing logging protocols enforce a number of security requirements. However, none of the published protocols offer evidentiary standards satisfying the admissibility requirements.

VI. CONCLUSION

While log data are being increasingly used as digital evidence in judicial disputes, we have shown that protocols used to collect and store log data fail to satisfy the security and hence legal requirements for admissible evidence. In consequence, more often than not, digital evidence based on log data can be successfully challenged in the court, leading to inadmissibility or loss on probative force. Intensive research into logging protocols is needed to advance digital evidence, in particular into the elucidation of further, fine-grained security requirements and into the verification of protocols with regard to these requirements. As further work, we intend to address these issues and investigate the trustworthiness of log data, a relevant topic neglected in our analysis for space contraints.

REFERENCES

[1] R. Accorsi, "On the relationship of privacy and secure remote logging in dynamic systems," in IFIP *Security and Privacy in Dynamic Environments*, S. Fischer-Hübner et al., Eds., 2006, vol. 201, pp. 329–339.

[2] ——, "Automated counterexample-driven audits of authentic system records," Ph.D. dissertation, University of Freiburg, 2008.

[3] R. Accorsi and A. Hohl, "Delegating secure logging in pervasive computing systems," in *Security in Pervasive Computing*, ser. LNCS, J. Clark et al., Eds., 2006, vol. 3934, pp. 58–72.

[4] M. Alles, A. Kogan, and M. Vasarhelyi, "Black box logging and tertiary monitoring of continuous assurance systems," *Inf. Sys. Cont.*, 2003.

[5] M. Bellare and B. Yee, "Forward integrity for secure audit logs," UCSD, Dept. of Computer Science & Engineering, Tech. Rep., 1997.

[6] B. Bloom, "Space/time trade-offs in hash coding with allowable errors," *Comm. ACM*, vol. 13, no. 7, pp. 422–426, 1970.

[7] D. Boneh and M. Franklin, "Identity based encryption from the Weil pairing," *SIAM J. of Comp.*, vol. 32, no. 3, pp. 586–615, 2003.

[8] C. Chong, Z. Peng, and P. Hartel, "Secure audit logging with tamper-resistant hardware," in IFIP *Security and Privacy in the Age of Uncertainty*, D. Gritzalis et al., Eds., vol. 250. 2003, pp. 73–84.

[9] J. Clark and J. Jacob, "A survey of authentication protocol literature," 1997, available at http://www.cs.york.ac.uk/~jac/papers/drareview.ps.gz.

[10] F. Cohen, *Challenges to Digital Evidence*. ASP Press, 2008.

[11] D. Dolev and A. Yao, "On the security of public key protocols," *IEEE Trans. Inf. Theo.*, vol. 2, no. 29, pp. 198–208, 1983.

[12] M. Franklin, "A survey of key evolving cryptosystems," *Journal of Security and Network*, vol. 1, no. 1/2, pp. 46–53, 2006.

[13] J. Holt, "Logcrypt: Forward security and public verification for secure audit logs," in *Symp. Grid Computing and e-Research*, R. Buyya et al., Eds., vol. 54., 2006, pp. 203–211.

[14] Kahn Consulting, "Computer security log files as evidence," http://www.kahnconsultinginc.com/images/pdfs/KCI_ArcSight_ESM_Evaluation.pdf, 2006.

[15] J. Kelsey and J. Callas, "Signed syslog messages," IETF Internet Draft, 2005, http://www.ietf.org/internet-drafts/draft-ietf-syslog-sign-16.txt.

[16] E. Kenneally, "Digital logs – Proof matters," *Digital Investigation*, vol. 1, no. 2, pp. 94–101, 2004.

[17] K. Kent and M. Souppaya, *Guide to Computer Security Log Management*, NIST, September 2006.

[18] L. Lamport, "Password authentication with insecure communication," *Comm. ACM*, vol. 24, no. 11, pp. 770–772, 1981.

[19] C. Lonvick, "RFC 3164: The BSD syslog protocol," 2001, http://www.ietf.org/rfc/rfc3164.txt.

[20] D. Ma, "Practical forward secure sequential aggregate signatures," in *ACM ASIACCS*, M. Abe and V. D. Gligor, Eds., 2008, pp. 341–352.

[21] D. Ma and G. Tsudik, "A new approach to secure logging," *ACM TOS*, vol. 5, no. 1, pp. 1–21, 2009.

[22] U. Maurer, "New approaches to digital evidence," *Proc. IEEE*, vol. 92, no. 6, pp. 933–947, 2004.

[23] D. New and M. Rose, "RFC 3195: Reliable delivery for syslog," Request for Comments, 2001, http://www.ietf.org/rfc/rfc3195.txt.

[24] Y. Ohtaki, "Partial disclosure of searchable encrypted data with support for boolean queries," in *Advances in Policy Enforcement*, S. Jakoubi et al., Eds. 2008, pp. 1083–1090.

[25] R. Oppliger and R. Ritz, "Digital evidence: Dream and reality," *IEEE Security and Privacy*, vol. 1, no. 5, pp. 44–48, 2003.

[26] "Reliable syslog," http://security.sdsc.edu/software/sdsc-syslog/.

[27] B. Schneier and J. Kelsey, "Security audit logs to support computer forensics," *ACM TISSEC*, vol. 2, no. 2, pp. 159–176, 1999.

[28] V. Stathopoulos, P. Kotzanikolaou, and E. Magkos, "A framework for secure and verifiable logging in public communication networks," in *Critical Information Infrastructures Security*, ser. LNCS, J. Lopez, Ed., 2006, vol. 4347, pp. 273–284.

[29] "Syslog-ng web site," http://www.balabit.com/products/syslog_ng/.

[30] B. Waters, D. Balfanz, G. Durfee, and D. Smetters, "Building an encrypted and searchable audit log," in *11th Annual Network and Distributed System Security Symposium*, 2004.