The Meaning of Logs ⋆

Sandro Etalle[1,2] ⋆⋆, Fabio Massacci[1] and Artsiom Yautsiukhin[1]
sandro.etalle@utwente.nl and {evtiukhi, massacci}@dit.unitn.it

[1]University of Trento, DIT
[2]University of Twente, DIES

Abstract. While logging events is becoming increasingly common in computing, in communication and in collaborative work, log systems need to satisfy increasingly challenging (if not conflicting) requirements. Despite the growing pervasiveness of log systems, to date there is no high-level framework which allows one to model a log system and to check whether it meets the requirements it should satisfy.
In this paper we propose a high-level framework for modeling log systems, and reasoning about them. This framework allows one to give a high-level representation of a log system an to check whether it satisfies given audit and privacy properties which in turn can be expressed in standard logic. In particular, the framework can be used for comparing and assessing log systems. We validate our proposal by formalizing a number of standard log properties and by using it to review a number of existing systems.

1 Introduction

In the past few years we have witnessed a struggle between two competing forces: privacy protection and fight against cyber-crime. Privacy protection has called for new regulations [14, 5], new technological solutions [2, 4] and re-thinking of business interactions [8]. On the other hand, efforts in countering cyber-crime, have led to increasingly invasive laws [12] and new auditing and monitoring techniques [23, 3, 1].

Such clash is most evident in the realm of auditing in general, and in the regulations on how logs should be taken, maintained and deleted in particular. A folklore pun well describes the problem as follows: if logs mention private information they are forbidden and if they do not - they are useless. For instance, an important privacy requirement for log systems is the compliance with the maximal retention period (the time after which a company has to delete user's data) which in some cases must be determined on a need basis [2, 4, 11] (e.g. service providers have to delete logged data when they do not need it any longer to offer their services). On the other hand, logs have to be kept for audit purpose or for computer forensics. This problem goes beyond privacy in databases: Internet

⋆ This work was partly supported by the project EU-IST-IP-SERENITY, contract N 27587

⋆⋆ Permanent address: University of Twente, the first author's contribution took place when he was visiting the University of Trento.

Service Providers (ISPs) have similar regulations [10, 26]. A recent amendment to EU Directive N 2002/58/EC [12] requires service providers (i.e. ISPs, e-mail services, communication providers) to store their logs for not less than 6 months to help law enforcement agencies. Consequently, sensitive information about a user may be in the system after the user's own account has been deleted.

We notice that even though logs are ubiquitous in computing and telecommunication security and there is a significant amount of work on *analyzing logs*[1], we find relatively few papers on *design and analysis of log systems* [27, 21, 13] and on what security properties a log system may or should exhibit [16]. This is somehow striking in comparison with the large body of work on security properties for e.g. security protocols or security models for access control.

In this paper we define a formal framework for modeling and analyzing log systems, which allows one to provide a high level specification of a log system, thereby allowing her to check whether it has the expected properties (e.g., if it meets given privacy or audit requirements). in particular, our framework can be used to compare different log systems with each other.

To validate our proposal, we include a survey of the requirements that are applicable to log systems, and we show how to represent them formally. In addition, we have considered a number of log systems taken from the literature and we show how they compare to each other when modeled in our framework.

2 Log Requirements

To be useful, logs often have to meet various requirements. Here we list the most common of them (collected from various papers in the literature: ISO17799 [15], CC [16], [6]); later, we will be able to give a precise formalization of these properties. First we need to specify some notation: here we talk about (real world) *events* and call *trace* a sequence of events. In turn, a trace may be logged in a *log*; by *recovering a trace* we indicate the action of associating to a given log the trace(s) of event that could have generated it.

- *Completeness*: All events in a trace of events can be recovered from its log.
- *Partial Completeness*: All events in a trace of events matching a given property (relevant events) can be recovered from its log.
- *Past Independence*: In a trace, older events have no influence on the log and recovery of newer events.
- *Future Independence*: In a trace, newer events have no influence on the log entries of older events, nor on their recovery.
- *Context Independence*: The conjunction of past and future independence.
- *Chaining*: Valid logs become invalid if an intermediate record is altered.
- *Exactness*: The recovery of a log of a trace is unambiguous: given a log there is a unique trace of events which could have generated it.

[1] See the RAID conference series, for example.

Events in a trace usually have attributes (e.g. date, user name, address); the following properties concern whether a given log system allows or not to recover a certain attributes. This is particularly important for privacy protection.

- *Complete Anonymity (w.r.t. attribute A)* : The recovery of an event does not give any information on the value of its attribute A.
- *Ambiguity (w.r.t. attribute A)*: The recovery of an event does not allow one to establish the value of its attribute A.
- *Linkability (w.r.t. attribute A)*: It is possible to determine whether two recovered events had the same value for attribute A (notice that the system could still be ambiguous w.r.t. A).
- *Positive/Negative Monotonicity*: Newer events do not introduce/reduce anonymity in older events.

An example of a log system which is not past independent is e.g. Linux, which records a user's name together with the assigned pseudonym (note that in Linux pseudonyms are used for convenience, and not to preserve users' privacy). An example of a system which does not satisfy future independence is one in which log entries are destroyed after a given retention time. Positive monotonicity is important when we do not want to lose information we logged. Negative monotonicity is important from the privacy perspective.

3 A formal model of logs

To introduce our framework we start by providing the definition of the world model, which is the environment where logging takes place. Here and in the sequel, given a set X we denote by 2^X its powerset and by X^* the set of sequences of elements from X.

Definition 1. *A* World Model *is a tuple $\langle E, T, AD, \{AF_i\}_{i\in I}\rangle$; where: E is a set of real world events; $T \subseteq 2^{E^*}$ is a set of valid traces; AD is a general attribute domain which includes all possible dimensions (e.g. strings, real, data, etc.); $\{AF_i\}_{i\in I}$ is a set of attribute functions, which given a sequence of events return the corresponding sequence of attribute values: $E^* \mapsto (2^{AD})^*$ (e.g. user(), date()).*

Now we can define a log system which records events from the world model.

Definition 2. *Let WM=$\langle E, T, AD, \{AF_i\}_{i\in I}\rangle$ be a world model, then a* Log System *for WM is a tuple $\langle R, L, Log(), Rec()\rangle$; where: R is a set of records; $L \subseteq 2^{R^*}$ is a set of valid logs in the system.* Log*: $E^* \mapsto R^*$ is the function mapping a trace of events into a log.* Rec*: $R^* \mapsto 2^{E^*}$ is the function which given a log returns the set of traces with the log.*

In other words, the recovery function *Rec()* maps a log into the set of traces of events that could have originated the log. Considering that some information might be lost during the logging process (e.g., in the case of anonymous systems),

it can well be the case that the *Rec*(l) contains more than one trace. We denote events by e and records by r. A trace is represented by $t = \langle e_1, e_2 ... e_n \rangle$. Similarly, a log is denoted by $l = \langle r_1, r_2 ... r_n \rangle$. In the sequel, $x \circ x'$ means that sequence x' is appended to (after) sequence x preserving elements order.

Example. Let us describe a log system using pseudonyms (as in [17]). Consider a hospital-based database containing medical and personal data of patients. The hospital keeps track of all accesses to the database both to prevent data linkage (privacy) and for accountability purposes. To define the World Model, we introduce the following domains: *Time* is a set of positive integer values which denote time; *Operator* is a set of users (represented as a strings) who have access to patient data; *Patient* is a set of all possible patients of the hospital (represented as strings); *Status* is the set: {successful, failed} used to denote if an action was accomplished by the system or not. The general attribute-domain is AD=*Time* $\bigcup$ *Operator* $\bigcup$ *Patient* $\bigcup$ *Status*. Finally, attribute functions are defined and named according to as the domains above AF={*Time(), Operator(), Patient(), Status(), Data()*}. In the world model, there are six types of events (here, $t \in Time$; $o \in Operator$; $p \in Patient$; $s \in Status$):

E={	*login(t,o,s)*	(Operator) o logged-in at time t;
	logoff(t,o,s)	o logged-off at time t;
	add(t,o,p,s)	o added the record of p to the system at time t;
	read(t,o,p,s)	o read the record of p at time t;
	update(t,o,p,s)	o updated the record of p at time t;
	delete(t,o,p,s)	o deleted p from the system at time t}

Having defined the possible events, a valid trace is any ordered (in time) sequence of such events. $T = \{t \in E^* |$ if $\forall e_i \in t \ \wedge \ \forall e_j \in t \ . \ i < j \implies Time(e_i) < Time(e_j)\}$. We can now move on to the definition of the log system. Let us first define some additional domains: *Patient_id* is a set of all possible identifiers (strings) of all patients; *Record_id* is a set of integers which unambiguously point at a log record. Note, that the *Patient* domain from the world model differs from *Patient_id*, as the real names of patients are substituted with pseudonyms. We underline the identifier to refer to the pseudonym, so $\underline{p}$ is the pseudonym of patient p. We also underline the records to distinguish between records and events. The log system has four types of records (here,: $j \in Record\_id$; $t \in Time$; $o \in Operator$; $\underline{p} \in Patient\_id$; $s \in Status$;):

R={	*add*$(j,t,o,\underline{p},s)$	o added the record of $\underline{p}$ to the system at time t;
	read$(j,t,o,\underline{p},s)$	o read record of $\underline{p}$ at time t;
	update$(j,t,o,\underline{p},s)$	o changed record of $\underline{p}$ at time t;
	delete$(j,t,o,\underline{p},s)$	o deleted $\underline{p}$ from the system at time t }

Record identifiers (j) are assigned incrementally. We can now define the *Log()* function:

$$Log(t) = \begin{cases} \underline{add}(j,t,o,\underline{p},s) \circ Log(t') \text{ if } t = e \circ t' \text{ and } e = add(t,o,p,s); \\ \underline{read}(j,t,o,\underline{p},s) \circ Log(t') \text{ if } t = e \circ t' \text{ and } e = read(t,o,p,s); \\ \underline{update}(j,t,o,\underline{p},s) \circ Log(t') \text{ if } t = e \circ t' \text{ and } e = update(t,o,p,s); \\ \underline{delete}(j,t,o,\underline{p},s) \circ Log(t') \text{ if } t = e \circ t' \text{ and } e = delete(t,o,p,s); \\ Log(t') \text{ if } t = e \circ t' \text{ and none of the above applies;} \\ \varepsilon \text{ otherwise.} \end{cases}$$

The mapping between a patient and his pseudonym is done with a special binding table to which access is restricted. Notice that we assume that *login* and *logoff* events are not logged. To define the recovery function, let M be the set of bijective mappings $Patient\_id \mapsto Patient$; given $m \in M$ we define R_m as follows:

$R_m(l \circ \underline{add}(j,t,o,\underline{p},s)) = R_m(l) \circ add(j,t,o,m(\underline{p}),s)$
$R_m(l \circ \underline{read}(j,t,o,\underline{p},s)) = R_m(l) \circ read(j,t,o,m(\underline{p}),s)$
$R_m(l \circ \underline{update}(j,t,o,\underline{p},s)) = R_m(l) \circ update(j,t,o,m(\underline{p}),s)$
$R_m(l \circ \underline{delete}(j,t,o,\underline{p},s)) = R_m(l) \circ delete(j,t,o,m(\underline{p}),s)$

(where $R_m(\varepsilon) = \varepsilon$); then we can define the recovery function: $Rec(l) = \{t |\ t = R_m(l) \text{ for some } m \in M\}$. Notice that the recovery function maps a log into a set of traces. Consider the following list of events: *Login(8:58 21/10/2006,*[2]*,Edward Green,successful) Add(10:30 21/10/2006,Edward Green,Mackle Daniels,successful) Login(12:00 21/10/2006,Suzi Wallach,successful) Changed(12:21 21/10/2006,Suzi Wallach,Paul Anderson,failed) Changed(12:22 21/10/2006,Suzi Wallach,Mackle Daniels,successful)* Then the corresponding log is:

Record ID	**Cause**	**Time**	**Operator**	**Patient**	**Status**
1	Add	10:30 21/10/2006	Edward Green	102	successful
2	Update	12:21 21/10/2006	Suzi Wallach	101	failed
3	Update	12:22 21/10/2006	Suzi Wallach	102	successful

As one can see the log file itself (without knowledge of the bijection mapping) does not disclose any information about the patients of the hospital other than the fact that records 1 and 3 concern the same patient. If an operator who has no access to the private data tries to recover the log he obtains six possible traces: one for each pseudonym-user assignment.

4 Properties

The formal log system allows us to give a precise definition of the informal properties stated in Section 2, this will provide us with a basis for assessing and comparing different log systems.

Having a formal definition of these properties is very important to make them precise, which is a less trivial task than it may seem at first. For instance, even a simple property such as *Completeness* may be defined in different ways; in particular, if we allow new events to cause the deletion of past records (which is

[2] Time is stored as an integer value, but for the sake of simplicity it is represented as usual.

usual in the real world) the definition changes. Let WM=$\langle E, T, AD, \{AF_i\}_{i\in I}\rangle$ be a world model, and $\langle R, L, Log(), Rec()\rangle$ be a log system for WM:

Definition 3 (Properties).

- *Trace Completeness:* $\forall t \in T\ t \in Rec(Log(t))$.
- *Partial Trace Completeness (w.r.t. a property P. Here we simply indicate by $P(t)$ the subsequence of t consisting of all and only events satisfying property P).* $\forall t \in T\ .\ P(t) \in Rec(Log(t))$.
- *Future Independence:* $\forall t, t' \in T\ .\ t' \in Rec(Log(t)) \iff \forall t_1 \in T\ \exists t'_1 \in T\ .\ t' \circ t'_1 \in Rec(Log(t \circ t_1))$
- *Past Independence:* $\forall t, t' \in T\ .\ t' \in Rec(Log(t)) \iff \forall t_1 \in T\ \exists t'_1 \in T\ .\ t'_1 \circ t' \in Rec(Log(t_1 \circ t))$.
- *Context Independence: conjunction of future and past independence.*
- *Chaining:* $l \circ \langle r\rangle \circ l' \in L \implies \forall r' \neq r \quad l \circ \langle r'\rangle \circ l' \notin L$.
- *Exactness:* $\forall t \in T\ \{t\} = Rec(Log(t))$

To express most privacy-related properties we need to be able to make the correspondence between single events and single log entries. In particular, if e is an event in a trace t and $t' \in Rec(Log(t))$ we have to be able to tell which event in t' corresponds to the original e. We denote this event by $t' \downarrow e$. In most cases, the correspondence function $\downarrow$ is realized quite simply by assigning consecutive numbers to events and log entries.

Definition 4 (Privacy Properties). *Let AF be an attribute function.*

- *Complete Anonymity (w.r.t. AF):* $\forall t \in T\ \forall t' \in Rec(Log(t))\ \forall e_1, e_2 \in t', AF(e_1) = AF(e_2)$.
- *Ambiguity (w.r.t. AF):* $\forall t \in T\ \forall e \in t\ |AF(Rec(Log(t)) \downarrow e)| > 1$.
- *Linkability (w.r.t. AF):* $\forall t \in T\ \forall e_i, e_j \in t\ .\ AF(Rec(Log(t)) \downarrow e_i) = AF(Rec(Log(t)) \downarrow e_j) \iff AF(e_i) = AF(e_j)$
- *Positive Monotonicity (w.r.t. AF):* $\forall t, t' \in T\ \forall e \in t\ AF(Rec(Log(t)) \downarrow e) \subseteq AF(Rec(Log(t \circ t')) \downarrow e)$
- *Negative Monotonicity (w.r.t. AF):* $\forall t, t' \in T\ \forall e \in t\ AF(Rec(Log(t)) \downarrow e) \supseteq AF(Rec(Log(t \circ t')) \downarrow e)$

Consider again the system shown in the Section 3. The system is not complete since exist events (e.g., t'=*Login(t,o,s)*) that are not logged corresponding logs; it is partially complete w.r.t. the property P which is true for all events except for *login* and *logoff*. The system in our example is context independent because the recovery of a record does not depend on other records; it is not chained since changes in the log are not noticeable by the system since by definition of L any sequence of records from R is valid; it is not exact because pseudonyms are mapped back (by the recovery function) to any person belonging to the set of patients; for the same reason, it is completely anonymous. Notice however that the system is still linkable: it allows us to see if two events pertain to the same patient (though the presence of the pseudonym does not allow to see which patient it is) as every user has only one identifier and visa versa. It is monotonic since it is context independent (see Theorem 1).

We can now relate some of these properties to each other. The proof of the following theorem is given in the Appendix.

Theorem 1.

1. *Every exact system is complete.*
2. *Every ambiguous system is not exact.*
3. *Every exact system is context independent.*
4. *Every exact system is (positively and negatively) monotonic.*
5. *Every future independent system is (positively and negatively) monotonic.*
6. *Every complete and anonymous system is ambiguous.*

Examples. We now use the properties just defined to assess and compare some existing logging systems.

Pseudonyms based systems [17–19]. These systems use pseudonyms to hide user identities to regular log users while allowing special authorized parties (who have access to the pseudonymization function) carry out precise auditing, The basic idea is to substitute private information with an arbitrary string (the pseudonym). The correspondence between pseudonyms and user identifiers is stored in some binding database with restricted access (here we should mention that even the use of pseudonyms does not guarantee complete protection from the use of statistical methods to reconstruct the behavior of user [9, 19]).

Linux logs. Sometimes pseudonyms are used for convenience rather than for privacy reasons. The Linux log system is an example of such pseudonymization ante-literam: here, user identities are partially hidden using group and user identifiers which can be considered as pseudonyms (e.g. user "grayyoga" has pseudonym 1001:100). The binding between user and her pseudonym is stored in the */etc/passwd* file. Still, all kernel audit information contains no references to the username but only to the user id[3]. On the other hand, when a new user is added in the system his identity and pseudonym are stored in a log file. This means that even if access to the */etc/passwd* file is denied it is possible to recover a user identity by consulting the log file.

Buschkes-Kesdogan system [7]. Buschkes and Kesdogan proposes a log system which uses group pseudonyms (e.g. for access right management). Before logging into a server a user receives from a Trusted Third Party (TTP) a credential containing a Group Reference Pseudonym (GRP). When she connects to the server, she reveals her credential and the server authenticates the user as a member of a group according to the GRP. The main peculiarity of the log system is that a pseudonym id corresponds to a *set* of user identifiers.

IDA system [24]. The IDA log system uses a pseudonyms as well, but instead of substituting the private information with a pseudonym the data is encrypted. The main advantage of the system is that for re-identification of logs no binding database is required: the information needed for full recovery is a decryption key.

[3] See `http://www.die.net/doc/linux/man/man8/auditctl.8.html`

Waters et al. [27]. Our last example is to the system of Waters et al., where log entries encrypted as a whole. This solution eliminates the need to store the correspondence between users and their pseudonyms. The disadvantage of this approach is that – in general – searching in encrypted logs is difficult. To overcome this problem the system stores keywords of each log entry which in turn are encrypted with a unique key. This allows the system to carry out limited search actions while offering good data protection. The authors also use a hash function to preserve the order and integrity of the log.

Finally, we can compare these systems in terms of some of the properties we have defined above.

	Context Independence	Complete Anonymity	Ambiguity	Monotonicity	Linkability	Exactness [4]	Chaining
Linux logs	–	–	–	√	√	–	–
Lundin–Johnnson [17]	√	√	√	√	√	–	–
Buschkes–Kesdogan [7]	√	–	√	√	–	–	–
IDA [24]	√	√	√	√	√	–	–
Waters et al. [27]	√	√	√	√	–	–	√

5 Conclusion and Related Works

In this paper we propose an abstract framework for formalizing and reasoning about log systems. Our framework allows one to model a concrete log system and to check whether it satisfies certain properties; in particular it allows one to check whether the system meets the various requirements such as the one we have collected from the literature [6, 16, 24, 7].

The practical motivation for realizing this framework is given by the need compare and assess precisely different log systems against the properties they have to satisfy, which in our system can be expressed in a precise way using a simple logic formalism. To validate our framework, we have encoded in it a number of different systems ([17, 7, 24, 27]). To the best of our knowledge, this is the first framework of this kind.

Related works There are a few works focusing on audit log properties. Billable and Yee in [6] introduce forward integrity property and propose a system enforcing it. The well-known Common Criteria security standard [16] reports some

[4] Note, that all these systems except [7] become exact if the operator has access to the inverse mapping from pseudonyms to user names. Also note, that in contrast to the system shown in our example, the system in [17] is complete and, therefore, exact if the above condition holds.

requirements which we have referred to as properties. The audit requirements specify what should be stored in the logs and how to use the logs collected.

Most log formalizations have been developed for monitoring purposes [22, 20, 25]. Roger and Goubauld-Larrecq [22] investigate linear time logic for log auditing and propose another logic consisting of Wolper-style linear-time formulae which make auditing more efficient. Spanoudakis et al. [25] propose a formal description of compliance checking for web-service based systems. Mansouri-Samani and Sloman [20] present GEM (generalized event monitoring language) which is used for monitoring networks and distributed systems. B. Waters et al. [27] provide a formal description of a searchable temper-resistant log model.

References

1. R. Agrawal, R. J. Bayardo, C. Faloutsos, J. Kiernan, R. Rantzau, and R. Srikant. Auditing compliance with a hippocratic database. In M. A. Nascimento, M. T. Özsu, D. Kossmann, R. J. Miller, J. A. Blakeley, and K. B. Schiefer, editors, *Proceedings of the 30th International Conference on Very Large Data Bases (VLDB'04)*, pages 516–527. Morgan Kaufmann, 2004.
2. R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu. Hippocratic databases. In P. A. Bernstein, Y. E. Ioannidis, R. Ramakrishnan, and D. Papadias, editors, *Proceedings of the 28th International Conference on Very Large Data Bases (VLDB'02)*. Morgan Kaufmann, 2002.
3. J. Allen, A. Christie, W. Fithen, J. McHugh, J. Pickel, and E. Stoner. State of the practice of intrusion detection technologies. Technical Report CMU/SEI-99-TR-028, Carnegie Mellon, SEI, January 2000.
4. P. Ashley, S. Hada, G. Karjoth, C. Powers, and M. Schunter. Enterprise privacy authorization language (EPAL 1.1). Technical report, IBM, October 2003. available via `http://www.zurich.ibm.com/security/enterprise-privacy/epal/` on 31/05/2005.
5. D. L. Baumer, J. B. Earp, and J. Poindexter. Internet privacy law: a comparison between the united states and the european union. *Computers & Security*, 23(5):400–412, July 2004.
6. M. Bellare and B. Yee. Forward integrity for secure audit logs. Technical report, University of California at San Diego, 1997.
7. R. Buschkes and D. Kesdogan. Privacy enhanced intrusion detection. In G. Mueller and K. Rannenberg, editors, *Proceedings of the Conference on Multilateral Security for Global Communication*, pages 187–207. Addison-Wesley-Longman, 1999.
8. L. Cranor, M. Langheinrich, M. Marchiori, M. Presler-Marshall, and J. Reagle. *The Platform for Privacy Preferences 1.0 (P3P1.0) Specification.* W3C, 1.0 edition, April 2002.
9. J. Domingo-Ferrer and V. Torra. Disclosure risk assessment in statistical microdata protection via advanced record linkage. *Statistics and Computing*, 13:343–354, 2003.
10. Electronic Privacy Information Center. Data retention. available via `http://www.epic.org/privacy/intl/data%5Fretention.html` on 30/12/2005, 2005.
11. EU. Directive 95/46/EC of the european parliament and of the council. available via Euro-Lex site `http://europa.eu.int/eur-lex` on 30/12/2005, 1995.

12. EU. DIRECTIVE 2006/24/EC OF THE EUROPEAN PARLIAMENT AND OF THE COUNCIL on the retention of data generated or processed in connection with the provision of publicly available electronic communications services or of public communications networks and amending directive 2002/58/ec. *Official Journal of the European Union*, 105/54, 2006. avialble via `http://www.ispai.ie/DR%20as%20published%20OJ%2013-04-06.pdf` on 21/06/2006.
13. J. V. Hansen. Audit considerations in distributed processing systems. *Communications of the ACM*, 26(8):562 – 569, August 1983.
14. HIPAA. Health insurance reform: Security standards; final rule. *Federal Register*, 68(34):8333–8381, February 2003. available via `http://www.hipaadvisory.com/regs/Regs%5Fin%5FPDF/finalsecurity.pdf` on 31/05/2005.
15. ISO/IEC. *Information technology Security techniques Evaluation criteria for IT security*, November 2001. available via `http://www.standardsdirect.org/iso17799.htm` on 01/06/2006.
16. ISO/IEC. *Common Criteria for Information Technology Security Evaluation*. Common Criteria Project Sponsoring Organisations, 2.2 edition, January 2004. available via `http://www.commoncriteriaportal.org/public/expert/index.php?menu=3` on 27/10/2005.
17. E. Lundin and E. Jonnson. Privacy vs. intrusion detection analysis. In *Proceedings of the 2nd International Symposium on Recent Advances in Intrusion Detection*, 1999. available via `http://www.raid-symposium.org/raid99/` on 28/02/2007.
18. E. Lundin and E. Jonsson. Anomaly-based intrusion detection: privacy concerns and other problems. *Computer Networks*, 34(4):623–640, October 2000.
19. B. Malin and L. Sweeney. How (Not) to Protect Genomic Data Privacy in a Distributed Network: Using Trail Re-identification to Evaluate and Design Anonymity Protection Systems. *Journal of Biomedical Informatics*, 37(3):179–192, 2004.
20. M. Mansouri-Samani and M. Sloman. GEM: a generalized event monitoring language for distributed systems. *Distributed Systems Engineering Journal*, 4:96–108, 1995.
21. Y. Ohtaki, M. Kamada, and K. Kurosawa. A scheme for partial disclosure of transaction log. *IRICE Transaction on Fundamentals of Electronics Communications and Computer Sciences*, E88(1):222–229, January 2005.
22. M. Roger and J. Goubault-Larrecq. Log auditing through model checking. In D. C. Young, editor, *Proceedings of the 2001 IEEE Computer Society Security Foundations Workshop*, pages 220–236. IEEE Computer Society Press, 2001.
23. B. Schneier and J. Kelsy. Secure audit logs to support computer forensics. *ACM Transactions on Information and System Security*, 2(2):159 – 176, May 1999.
24. M. Sobirey, S. Fischer-Hoebner, and K. Rannenberg. Pseudonymous audit for privacy enhanced intrusion detection. In L. Yngstroem and J. Carlsen, editors, *Proceedings of the IFIP TC11 13 international conference on Information Security (SEC '97) on Information security in research and business*, pages 151–163, London, UK, UK, 1997. Chapman & Hall, Ltd.
25. G. Spanoudakis and K. Mahbub. Non intrusive monitoring of service based systems. *International Journal of Cooperative Information Systems*, 15(3):325–358, 2006.
26. Statewatch. UK-EU call for mandatory data retention of all telecommunications. available via `http://www.statewatch.org/news/2005/jul/05eu-data-retention.htm`, 2005.

27. B. R. Waters, D. Balfanz, G. Durfee, and D. K. Smetters. Building an encrypted and searchable audit log. In *Proceedings of the 11th Annual Symposium on Network and Distributed System Security*, San Diego, 2004. available via `http://www.isoc.org/isoc/conferences/ndss/04/proceedings/index.htm` on 28/02/2007.

A Appendix

We now report the proof of Theorem 1. **Note: This appendix is included solely for the reviewer's convenience. Should the paper be accepted the appendix will be removed from the paper; the proof it contains will be made available as technical report**.

1. *Every exact system is complete.*
   According to the definition of exactness, $\forall t \in T\ \{t\} = Rec(Log(t))$. This means $t \in Rec(Log(t))$ holds in this case. But this is the definition of completeness.
2. *Every ambiguous system is not exact.*
   According to the definition of ambiguity: $\forall t \in T\ \forall e \in t \mid AF_i(Rec(Log(t)) \downarrow e)| > 1$. This means that the event itself is ambiguous $|Rec(Log(t)) \downarrow e| > 1$, and so the whole trace $|Rec(Log(t))| > 1$. Because of this, we certainly have that $\{t\} = Rec(Log(t))$.
3. *Every exact system is context independent.*
   Consider three traces: $\forall t_1, t_2, t_3 \in T$ such that $t_2 = t \circ t_1$. Now apply the definition of exactness to these traces: $\{t_1\} = Rec(Log(t_1))$ and $Rec(Log(t_2)) = \{t_2\}$. According to our assumption that $t_2 = t \circ t_1$ the second equation may be rewritten as $Rec(Log(t \circ t_1)) = \{t \circ t_1\}$ This is the definition of past independence. The proof that an exact system is future independent can be done in the same way with the assumption that $t_2 = t_1 \circ t$.
4. *Every future independent system is (positively and negatively) monotonic.*
   According to the definition of future independence: $\forall t, t' \in T\ .\ t' \in Rec(Log(t)) \iff \forall t_1 \in T\ \exists t'_1 \in T\ .\ t' \circ t'_1 \in Rec(Log(t \circ t_1))$. Let us take two sets of events which are received after recovering event e from a trace: $E' = \{t' \downarrow e \mid t' \in Rec(Log(t))\}$ and $E'' = \{t'' \downarrow e \mid t'' \in Rec(Log(t \circ t_1))\}$ where $t'' = t' \circ t'_1$ and $e \in t'$. These two sets are equal since the parts of traces from which we took the events are identical by the definition of future independence. This means that: $AF(Rec(Log(t) \downarrow e)) = AF(Rec(Log(t \circ t_1) \downarrow e))$ and that the system is positive/negative monotonic.
5. *Every exact system is (positively and negatively) monotonic.*
   The proof follows from the facts that: exact systems are context independent, the context independent systems are future independent and the future independent systems are (positively and negatively) monotonic.
6. *Every complete and anonymous system is ambiguous* [5].
   Consider any two events $\forall e_1, e_2 \in E$ which belong to some trace $e_1, e_2 \in t$ and have different attribute values $AF(e_1) \neq AF(e_2)$. Since the system is complete $e_1 \in AF(Rec(Log(t)) \downarrow e_1)$ and $e_2 \in AF(Rec(Log(t)) \downarrow e_2)$. Then, according to the definition of complete anonymity : $AF(Rec(Log(t)) \downarrow e_1) = AF(Rec(Log(t)) \downarrow e_2) \supseteq \{e_1, e_2\}$. This implies that $|AF(Rec(Log(t)) \downarrow e)| > 1$

[5] We also assume that there is more than one user in the system since, otherwise, privacy protection does not make sense