Log Analysis for Data Protection Accountability

Denis Butin, Daniel Le Métayer

► To cite this version:

Denis Butin, Daniel Le Métayer. Log Analysis for Data Protection Accountability. FM2014 - 19th International Symposium on Formal Methods, Cliff Jones and Pekka Pihlajasaari and Jun Sun, May 2014, National University of Singapore (NUS), Singapore. pp.163-178, 10.1007/978-3-319-06410-9_12 . hal-00984308

HAL Id: hal-00984308
https://hal.inria.fr/hal-00984308

Submitted on 28 Apr 2014

Log Analysis for Data Protection Accountability

Denis Butin and Daniel Le Métayer

Inria, Université de Lyon, France
{denis.butin, daniel.le-metayer}@inria.fr

Abstract. Accountability is increasingly recognised as a cornerstone of data protection, notably in European regulation, but the term is frequently used in a vague sense. For accountability to bring tangible benefits, the expected properties of personal data handling logs (used as "accounts") and the assumptions regarding the logging process must be defined with accuracy. In this paper, we provide a formal framework for accountability and show the correctness of the log analysis with respect to abstract traces used to specify privacy policies. We also show that compliance with respect to data protection policies can be checked based on logs free of personal data, and describe the integration of our formal framework in a global accountability process.

1 Context and Motivation

The principle of accountability, introduced three decades ago in the OECD's guidelines [18], has been enjoying growing popularity over the last few years in the field of data protection. A consortium was set up in 2009 with precisely the definition and analysis of accountability as one of its primary goals [8]. At the European level, the Article 29 Working Group published an opinion dedicated to the matter recently [1] and the principle is expected to be enshrined in the upcoming European data protection regulation [12][1]

The key idea behind the notion of accountability is that data controllers (European terminology for entities collecting personal data, denoted DC from now on) should not merely comply with data protection rules but also be able to demonstrate compliance — "showing how responsibility is exercised and making this verifiable", as stated by the Article 29 Working Group [1]. The motivation underlying this general principle is that data subjects (DS) disclosing personal data to a DC lose control over it and require strong guarantees regarding actual handling.

Crucially, accountability is more than an impediment to companies: it can help them clarify their internal processes and level of compliance with legal rules (or their own policies). In addition, a solid accountability process puts a company in a better position to demonstrate its compliance in case of dispute.

[1] The latest draft of this regulation, adopted by the European Parliament's Civil Liberties Committee last October, further strengthens accountability requirements (articles 5 and 22).

Nevertheless, a downside to the generality of this concept is that it is too frequently used in a vague sense — at least, by lawyers and decision makers. Some clarity is provided by Bennett's nomenclature [9], which distinguishes three types of accountability: accountability of policy, of procedures and of practice. The strongest variant is accountability of practice, which holds that DC ought to demonstrate that their actual data handling complies with their obligations. In the case of accountability of policy, they must be able to show that they actually have defined a privacy policy; in the case of accountability of procedures, they must be able to show that they have put in place appropriate procedures to meet their policy. Ideally, the three types of accountability should be implemented: having a privacy policy in place is obviously a minimal requirement and the procedures should support good practices. However, in order to implement the accountability of practices and ensure that it can really improve the protection of DS, a number of key questions must be addressed:

- A clear definition should be provided of the "accounts" which are at the core of the concept of accountability. For accountability of practice, execution logs are natural candidates, but what should be kept in the logs is an essential and non-trivial issue. Obviously, enough information should be recorded to make accountability possible; but it is also necessary to comply with another principle of data protection, data minimization: only the personal data necessary for a given purpose should be recorded. Actually, one of the arguments against the use of accountability of practice is that the logs required to implement it could in fact represent an additional source of risks for personal data. As illustrated in our work [4], designing the contents of the logs is therefore far from obvious: intuitive solutions typically include too much data or omit information necessary for effective compliance.
- A clear definition of the accountability process has to be provided, showing how accounts are built and analyzed. For the accountability process to be worthwhile, accounts (here: logs) should reflect actual system execution and the verdict returned by the analysis procedure ought to be reliable. Overall, the guarantees provided by the whole process should be detailed to avoid misleading representations by DC or misplaced expectations from DS.

If the above issues are not properly handled, accountability may either represent illusory protections (and low-cost greenwashing for DC) or even additional sources of personal data leaks.

In this paper, we argue that formal methods can play a crucial role in addressing the above issues. In this context, however, they have to be used in a "light" way for several reasons. First, not all data protection obligations can be described formally. For instance, the notion of purpose, which is central in the European Data Protection Directive, cannot be defined in mathematical terms. Similarly, *break-glass* rules [16], which are necessary in certain areas such as health data processing (e.g. to allow unauthorized physicians to access personal data in emergency situations), are not well-suited to formalisation. Furthermore, the goal of the accountability process is not to establish a formal proof of compliance for a system (which would be completely out of reach in practice) but rather to be able to detect potential misbehaviour. One challenge in this area is therefore the integration of formal methods in an otherwise informal process and the definition of clear interactions between both worlds.

Another issue to be addressed in a formal accountability framework is the gap between two different levels of abstraction. The privacy [2] policy defined or understood by DS (or by lawyers) applies to abstract notions, such as "home address" or "health data", whereas actual logs typically include lower-level details such as system memory addresses or duplication of data.

Considering the above objectives and challenges, the contributions of this paper are threefold:

- We provide a framework for accountability of practice based on "privacy friendly" logs, showing that compliance with respect to data protection policies can be checked based on logs which do not contain any personal data.
- We show the correctness of the log analysis with respect to abstract traces that are used to specify privacy policies.
- We describe the integration of the formal framework in the overall accountability process and identify the complementary procedures and manual verifications that are necessary to complement the log analysis.

We first introduce privacy policies and their abstract representation (§2), before specifying "personal-data-free" logs (§3). The core accountability properties, i.e. the guarantees provided by the log analysis, are presented in §4. The integration of the formal framework in a global accountability process is outlined in §5. We then provide a survey of related work (§6), followed by an outline of future work and conclusive remarks (§7). An extended version of this paper is available in a technical report [6].

2 Privacy Policies and Abstract Events

The first stage of any data protection accountability process is the definition of privacy policies. In practice, a policy can be defined by the DC and accepted by the DS or result from a negotiation phase. In any case, it should comply with applicable laws. We do not consider the legal validity of the policies here nor their origin and assume that any personal data received by a DC is associated with a policy. The fact that the data is sent with a policy by the DS implies that she provides her consent for the use of her data in the conditions expressed by the policy. The fact that the DC accepts the data with the policy is taken as a commitment from his side to comply with the policy. In practice, a policy specifies what can be done with categories of data defined in a way which makes sense to DS, for instance "age", "postal address", or "profession". A first and major requirement of our accountability framework is that the privacy policy

[2] In this paper, we use the expressions "privacy" and "data protection" interchangeably even though, from a legal point of view, they refer to two different protection regimes.

should always remain attached to the associated data (which is sometimes called the *sticky policy* approach) because it will serve as a reference point for evaluating whether the DC has fulfilled his obligations.

As we want to check compliance with respect to privacy policies, we consider traces and logs on the side of the DC in this paper.

Definition 1 (Privacy policy). *Privacy policies are defined as tuples:*

$$Policy = Purposes \times Time \times Time \times Contexts \times FwPolicy$$

In $\pi \in Policy, \pi = (ap, dd, rd, cx, fw)$, ap is the set of authorised purposes of data use. Purposes are taken from a set of admissible values (taken as constants here, possibly structured as an ontology). The deletion delay dd is the delay after which the data must be deleted by the DC. The rd parameter specifies the delay for the DC to comply with requests by the DS, for instance regarding the deletion of personal data. The set cx defines the contexts in which the data can be used. $Contexts$ is the set of constants here which could represent external parameters such as time or location. The data forwarding policy is defined by the value of fw; it is equal either to $\uparrow$ (in which case no forwarding at all to third parties is possible) or to $\downarrow$ (all forwarding is allowed). We sometimes use the notation $\pi.ap$, $\pi.dd$, etc. to access the fields of a policy tuple. An example policy in this format could be $\pi = (\{Marketing, Statistics\}, 180d, 60m, \{Location\_Europe\}, \uparrow)$. This policy stipulates that data can be used exclusively for the purposes of $Marketing$ and $Statistics$, that all data must be deleted no later than 180 days from its disclosure, that requests by the DS must be complied with within 60 minutes, that data can only be used for a location context equal to $Europe$ and that any forwarding to third parties is forbidden.

We do not attempt to include all complexities of existing policy languages here. The above format should rather be seen as a proof-of-concept example to illustrate our overall approach.

2.1 Abstract Events

Having defined privacy policies, we now introduce the list of abstract events, so-called because they describe events at the level of personal data, abstracting away from system internals such as memory addresses. Abstract events are expressed intuitively with regard to the format of privacy policies. Mirroring the design of privacy policies mentioned above, this list of events illustrates an instantiation of our framework; it can be extended easily[3]. All abstract events carry a timestamp t as their first argument.

- $(Disclosure, t, or, ds, \theta, v, \pi)$ — the initial reception by the DC of personal data of origin or (the origin is the entity which sent the data), type θ (e.g. a person's age or postal address) and value v related to DS ds, with an associated sticky policy π. Depending on the value of or, the data can be sent by ds or by a third party.

[3] For example with update events — one could add a modification index to states to manage them. Notifications events could also be added.

- $(\mathit{DeleteReq}, t, \mathit{or}, \mathit{ds}, \theta)$ — a request received by the DC and sent by or to delete personal data of owner ds and type θ.
- $(\mathit{AccessReq}, t, \mathit{ds}, \theta)$ — a request received by the DC and sent by ds to access her own data.
- $(\mathit{Delete}, t, \mathit{ds}, \theta)$ — a deletion of the data of ds of type θ by the DC.
- $(\mathit{DeleteOrder}, t, \mathit{tp}, \mathit{ds}, \theta)$ — a request sent by the DC to the third party tp to delete the data of ds of type θ.
- $(Forward, t, rec, \mathit{ds}, \theta, v, \pi)$ — the forwarding by the DC of the data of ds of type θ and value v to the recipient rec, which can be either a third party or the DS (to grant her access to her own data following an access request), with policy π attached.
- $(Use, t, \mathit{ds}, \theta, \mathit{purpose}, \mathit{reason})$ — the use by the DC of the data of ds of type θ for a specific *purpose* and *reason*. The *purpose* element is taken from an ontology, while the *reason* is a textual description, used by a human for informal verification as discussed in §5.
- $(\mathit{BreakGlass}, t, \mathit{et}, \mathit{bgt}, \mathit{bgc})$ — the occurrence of a break-glass event of type bgt in circumstances bgc, where the affected entities and data types are couples (ds, θ) members of the set et. In practice, bgc is a textual description, similarly to *reason* in Use events.
- $(\mathit{Context}, t, \mathit{ct})$ — the switching of the current context to ct. To simplify, the context is just modeled by a simple value here but it could very well be a structure to account for different external parameters (such as time, location, etc.).

Definition 2 (Trace). *A trace σ is a sequence of abstract events.*

In order to define the notion of compliant trace, we need to introduce abstract states.

Definition 3 (Abstract state). *The abstract state of a system is a function $S_A : \mathit{Entity} \times \mathit{Type} \longrightarrow \mathit{Time} \times \mathit{Entity} \times \mathit{Value} \times \mathit{Policy} \times \mathcal{P}(\mathit{Entity} \times \mathbb{N}) \times \mathcal{P}(\mathit{BGtype} \times \mathit{BGcircumstances} \times \mathit{Time})$*

$$(\mathit{ds}, \theta) \mapsto (t, \mathit{or}, v, \pi, \mathit{receivers}, \mathit{bg})$$

The abstract state associated with each DS ds and type of personal data θ includes the origin or (the entity from which the most recent version of the value of the data emanated from), the data's value v, the sticky policy π (current policy) and the set of *receivers* (all third parties who have received the data together with the corresponding event index in the trace). Information about break-glass events is collected by triples $\mathit{bg}_n = (\mathit{bgt}, \mathit{bgc}, \mathit{timebg})$, where bgt is a break-glass event's type, bgc its circumstances and timebg its time. bg is a set of such triples, including all break-glass events that occurred so far for this DS and data type. S_A is expanded with $S_A(\mathit{Context}) = \mathit{ct} \in \mathit{Context}$, where ct is the current context.

We use the notation $\Sigma[(\mathit{ds}, \theta) \to (t, \mathit{or}, v, \pi, r, \mathit{bg})]$ to denote a state Σ' similar to Σ except that $\Sigma'(\mathit{ds}, \theta) = (t, \mathit{or}, v, \pi, r, \mathit{bg})$. The semantics of an event at

$S_A\,((Disclosure, t, or, ds, \theta, v, \pi), j)\,\varSigma = \varSigma[(ds, \theta) \to (t, or, v, \pi, \varnothing, \varnothing)]$

$S_A((Delete, t, ds, \theta), j)\varSigma = \varSigma[(ds, \theta) \to \perp]$

$S_A((Forward, t', rec, ds, \theta, v, \pi), j)\varSigma =$
$\texttt{if}\ rec \neq ds\ \texttt{then}\ \varSigma[(ds, \theta) \to (t, or, v, \pi, receivers \cup \{(rec, j)\}, bg)]$
with $(t, or, v, \pi, receivers, bg) = \varSigma(ds, \theta)\quad \texttt{else}\ \varSigma$

$S_A((BreakGlass, t', et, bgt, bgc), j)\varSigma =$
$\texttt{if}\ (ds, \theta) \in et\ \texttt{then}\ \varSigma[(ds, \theta) \to (t, or, v, \pi, receivers, bg \cup \{(bgt, bgc, t')\})]$
with $(t, or, v, \pi, receivers, bg) = \varSigma(ds, \theta)\quad \texttt{else}\ \varSigma$

$S_A((Context, t, ct), j)\varSigma = \varSigma[Context \to ct]$

$S_A(\sigma_i, j)\varSigma = \varSigma$ for the other events. Even though those events do not impact the abstract state, they either introduce commitments for the DC (e.g. *DeleteReq*) or allow him to fulfill his obligations (e.g. *DeleteOrder*).

Fig. 1. Abstract event semantics

a given position j in a trace are given by the function S_A: $(Event \times \mathbb{N}) \to AbstractState \to AbstractState$ defined in Fig. 1.

Disclosure initialises all abstract state variables, while *Forward* adds a third party, together with its event index, to the *receivers* set, unless the recipient is the DS herself (i.e. the DS is granted access to her own data), in which case the state is unchanged. *BreakGlass* events only modify the state if they occur for the ds and θ under consideration.

The current state after the execution of a trace $\sigma = [e_1, \ldots, e_n]$ is defined as $F_A(\sigma, 1)\varSigma_0$ with $\forall\ ds, \theta,\ \varSigma_0(ds, \theta) = \perp$ and:

$$F_A\,([\ ], n)\,\varSigma = \varSigma$$

$$F_A\,([e_1, \ldots, e_m], n)\,\varSigma = F_A\,([e_2, \ldots, e_m], n+1)\,(S_A(e_1, n)\varSigma)$$

We set $State_A(\sigma, i) = F_A(\sigma_{|i}, 1)\varSigma_0$, with $\sigma_{|i} = \sigma_1 \ldots \sigma_i$ the prefix of length i of σ.

Furthermore, let $EvTime$ be a function such that $EvTime(\sigma_i) = t_i$ with $\sigma_i = (X, t_i, \ldots), t_i \in Time$. Having defined abstract events, traces and event semantics, we can now define the compliance of a trace with respect to the policy attached to the data received by a DC.

2.2 Trace Compliance Properties

The following compliance properties are stated $\forall\ i \in \mathbb{N},\ \forall\ ds,\ \forall\ \theta$:

A1: No personal data should appear in an abstract state after its global deletion delay has expired: $State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg) \implies EvTime(\sigma_i) \leq t + \pi.dd$

A2: Deletions yield third party deletion requests, sent between the last forwarding of the data and deletion: $\sigma_i = (Delete, t', ds, \theta) \ \wedge \ State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg) \implies \ \forall \ (t_p, l) \in receivers, \ \exists \ k \mid \exists \ t'' \mid \sigma_k = (DeleteOrder, t'', t_p, ds, \theta) \wedge k \in]\alpha, i[$ with $\alpha = max\{n \mid (t_p, n) \in receivers\}$

A3: Deletion requests are fulfilled before expiration of the request fulfillment delay: $\sigma_i = (DeleteReq, t', or, ds, \theta) \ \wedge \ State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg) \implies \ \exists \ k \mid \exists \ t'' \mid \sigma_k = (Delete, t'', ds, \theta) \ \wedge \ t' < t'' \leq t' + \pi.rd$

A4: A4 is defined similarly to A3 for access requests, where the granting of access is a $Forward$ event with $rec = ds$.

A5: Data is only used for purposes defined in the policy: $\sigma_i = (Use, t', ds, \theta, purpose, reason) \ \wedge \ State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg) \implies purpose \in \pi.ap$

A6: All contexts in which data is used in the trace are authorised in the policy: $\sigma_i = (Use, t', ds, \theta, purpose, reason) \ \wedge \ State_A(\sigma, i-1)(Context) = ct \ \wedge \ State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg) \implies ct \in \pi.cx$

A7: If the policy forbids all forwarding, there is none: $\sigma_i = (Forward, t', rec, ds, \theta, v, \pi) \ \wedge \ rec \neq ds \ \wedge \ State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg) \implies \pi.fw \neq \uparrow$

Definition 4 (Trace compliance). *A trace σ is compliant ($Compliant_A(\sigma)$) if it satisfies all of the above properties $A_1, \ldots, A_7$.*

This concludes our formalisation of abstract events. The next section introduces log events, which are closer to system operations and include internals such as memory references. Defining such events and their compliance will ultimately allow us to relate abstract events and log events to express accountability properties (§4).

3 Log Specification and Compliance

Abstract events are useful to express privacy policies at a level which makes sense for DS. However the expected guarantees concern the actual behaviour of the system, which can be checked based on its execution log. We start by defining log events and continue with the associated concrete states and compliance properties.

3.1 Log Events

There are two main differences between trace events and log events. First, log events correspond to a small number of general purpose low-level operations, such as receiving data, sending it, reading it, copying it, deleting it or external events. The semantics of these events are passed through parameters (in most cases, the second one, such as *Disclosure*). Second, log event operations apply to the machine state, which is a function from references (i.e. memory addresses) to values; as opposed to abstract event operations, which apply directly to high-level data.

The format of the logs is a key design choice for an accountability architecture. As discussed in [4], this choice is far from obvious. In our framework, it is guided by two factors: the privacy policies which have to be verified and the aforementioned data minimization principle. Actually, we choose a radical option here, which is to avoid recording in the logs any value v of personal data [4]. We show in the next section that this choice does not prevent us from meeting the expected accountability requirements.

The list of log events follows. All log events carry a timestamp t, and events without descriptions have the same meaning as the corresponding abstract event.

- $(\mathit{Receive}, \mathit{Disclosure}, t, \mathit{or}, \mathit{ds}, \theta, \pi, \mathit{ref})$
- $(\mathit{Receive}, \mathit{DeleteReq}, t, \mathit{or}, \mathit{ds}, \theta)$
- $(\mathit{Receive}, \mathit{AccessReq}, t, \mathit{ds}, \theta)$
- $(\mathit{Copy}, t, \mathit{ref}, \mathit{ref})$ — a copying of data by the DC from one system reference to another.
- $(\mathit{Delete}, t, \mathit{ref})$ — a deletion of the data of ds with reference ref by the DC.
- $(\mathit{Send}, \mathit{DeleteOrder}, t, \mathit{tp}, \mathit{ds}, \theta)$
- $(\mathit{Send}, \mathit{Val}, t, \mathit{rec}, \mathit{ref})$ — an unspecified sending of data from the DC to a recipient rec, which can be a third party or ds in case she is granted access to her own data.
- $(\mathit{Read}, t, \mathit{ref}, \mathit{purpose}, \mathit{reason})$ — the use by the DC of the data of ds of reference ref for a specific $\mathit{purpose}$ and reason.
- $(\mathit{External}, \mathit{BreakGlass}, t, \mathit{et}, \mathit{bgt}, \mathit{bgc})$
- $(\mathit{External}, \mathit{Context}, t, \mathit{ct})$

Logs are to traces as log events are to abstract events:

Definition 5 (Log). *A log is a sequence of log events.*

In the same way that we defined abstract states and semantics, we now define concrete states and the semantics of concrete events.

Definition 6 (Concrete state). *The concrete state of a system is defined by the function* $S_C : \mathit{Reference} \longrightarrow \mathit{Time} \times \mathit{Type} \times \mathit{Entity} \times \mathit{Entity} \times \mathit{Policy} \times \mathcal{P}(\mathit{Entity} \times \mathbb{N}) \times \mathcal{P}(\mathit{BGtype} \times \mathit{BGcircumstances} \times \mathit{Time})$

$$\mathit{ref} \mapsto (t, \theta, \mathit{ds}, \mathit{or}, \pi, \mathit{receivers}, \mathit{bg})$$

Here $\mathit{Reference}$ is the set of memory addresses; the other parameters are defined as for abstract states. S_C is expanded with $S_C(\mathit{Context}) = \mathit{ct} \in \mathit{Context}$.

The semantics of an event at a position j in a log are given by a function $(\mathit{LogEvent} \times \mathbb{N}) \rightarrow \mathit{ConcreteState} \rightarrow \mathit{ConcreteState}$ defined as in Fig. 2.

Note that data values are not manipulated explicitly here; e.g. in the concrete $(\mathit{Receive}, \mathit{Disclosure}, \ldots)$ event above, the value of the data of type θ is stored in system memory at address ref. The Copy event does not modify the state associated to ref but the one associated to ref', since ref' is overwritten.

[4] Nevertheless, the couple (ds, θ) to which v is associated is still recorded.

$S_C((\mathit{Receive}, \mathit{Disclosure}, t, or, ds, \theta, \pi, ref), j)\varSigma = \varSigma[ref \to (t, \theta, ds, or, \pi, \emptyset, \emptyset)]$

$S_C((\mathit{Copy}, t, ref, ref'), j)\varSigma = \varSigma[ref' \to \varSigma(ref)]$

$S_C((\mathit{Delete}, t, ref), j)\varSigma = \varSigma[ref \to \perp]$

$S_C((\mathit{Send}, Val, t', rec, ref), j)\varSigma =$
$\texttt{if}\ rec \neq ds\ \texttt{then}\ \varSigma[ref \to (t, \theta, ds, or, \pi, \mathit{receivers} \cup \{(rec, j)\}, bg)]$
with $(t, \theta, ds, or, \pi, \mathit{receivers}, bg) = \varSigma(ref)\quad \texttt{else}\ \varSigma$

$S_C((\mathit{External}, \mathit{BreakGlass}, t', et, bgt, bgc), j)\varSigma =$
$\texttt{if}\ (ds, \theta) \in et\ \texttt{then}\ \varSigma[ref \to (t, \theta, ds, or, \pi, \mathit{receivers}, bg \cup \{(bgt, bgc, t')\})]$
with $(t, \theta, ds, or, \pi, \mathit{receivers}, bg) = \varSigma(ref)\quad \texttt{else}\ \varSigma$

$S_C((\mathit{External}, \mathit{Context}, t, ct), j)\varSigma = \varSigma[\mathit{Context} \to ct]$

$S_C(L_i, j)\varSigma = \varSigma$ for the other events.

Fig. 2. Concrete event semantics

The current concrete state $\mathit{State}_C(L)$ after the execution of a log L is defined recursively from S_C, like $\mathit{State}_A(\sigma)$ was previously defined from S_A. One can now express useful functions based on the current state at a position i in a log:

- The *Locations* function returns the set of references associated to data of a certain datatype from ds:
  $\mathit{Locations}(L, i, ds, \theta) = \{ref \mid \mathit{State}_C(L, i)(ref) = (\_, \theta, ds, \_, \_, \_, \_)\}$
- The *AllReceivers* function returns the set of all third parties that store some data of a certain datatype from ds, with the associated event index at which they received the data: $\mathit{AllReceivers}(L, i, ds, \theta) = \{(t_p, k) \mid \exists\ ref \mid$ $\mathit{State}_C(L, i)(ref) = (\_, \theta, ds, \_, \_, \mathit{receivers}, \_)\ \wedge\ (t_p, k) \in \mathit{receivers}\}$

Furthermore, as for abstract events, let *EvTime* be a function such that $\mathit{EvTime}(L_i) = t_i$ when $L_i = (\ldots, t_i, \ldots)$. Using these functions, we can now express compliance for logs.

3.2 Log Compliance Properties

Because logs reflect actual system executions and involve lower-level operations such as copies of data in memory addresses, it is necessary to also define the meaning of compliance in terms of logs. The following log compliance properties are stated $\forall\ i \in \mathbb{N},\ \forall\ ref,\ \forall\ ds,\ \forall\ \theta$:

C1: No personal data should appear in an abstract state after its global deletion delay has expired: $\mathit{State}_C(L, i-1)(ref) = (t, \theta, ds, or, \pi, \mathit{receivers}, bg) \Longrightarrow$ $\mathit{EvTime}(L_i) \leq t + \pi.dd$

C2: Deletions yield third party deletion requests, sent between the last forwarding of the data and its deletion: $L_i = (Delete, t', ref) \ \wedge \ State_C(L, i-1)$ $(ref) = (t, \theta, ds, or, \pi, receivers, bg) \Longrightarrow \ \forall \ (t_p, l) \in receivers, \ \exists \ k \mid \exists \ t'' \mid$ $L_k = (Send, DeleteOrder, t'', t_p, ds, \theta) \ \wedge \ k \in]\alpha, i[$ with $\alpha = max\{n \mid (t_p, n)$ $\in receivers\}$

C3: Delete requests are fulfilled before expiration of the request fulfillment delay: $L_i = (Receive, DeleteReq, t', or, ds, \theta) \wedge State_C(L, i-1)(ref) = (t, \theta, ds, or,$ $\pi, receivers, bg) \Longrightarrow \ \forall \ r \in Locations(L, i, ds, \theta), \ \exists \ k \mid \exists \ t'' \mid L_k = (Delete,$ $t'', r) \ \wedge \ t' < t'' \leq t' + \pi.rd$

C4: C4 is defined similarly to C3 for access requests.

C5: Data is only used for purposes defined in the policy: $L_i = (Read, t', ref,$ $purpose, reason) \ \wedge \ State_C(L, i-1)(ref) = (t, \theta, ds, or, \pi, receivers, bg)$ $\Longrightarrow \ purpose \in \pi.ap$

C6: All contexts in which data is used in the trace are authorised in the policy: $L_i = (Read, t', ref, purpose, reason) \ \wedge \ State_C(L, i-1)(Context) = ct \ \wedge$ $State_C(L, i-1)(ref) = (t, \theta, ds, or, \pi, receivers, bg) \Longrightarrow \ ct \in \pi.cx$

C7: If the policy forbids all forwarding, there is none: $L_i = (Send, Val, t', rec, ref) \ \wedge \ rec \neq ds \ \wedge$ $State_C(L, i-1)(ref) = (t, \theta, ds, or, \pi, receivers, bg) \Longrightarrow \pi.fw \neq \uparrow$

Definition 7 (Log compliance). *A log L is compliant ($Compliant_C(L)$) if it satisfies all of the above properties $C_1, \ldots, C_7$.*

4 Accountability Properties

To relate abstract privacy policies to actual log verifications, it is necessary to introduce two abstraction relations: a relation between abstract states and concrete states and a relation between traces and logs.

We first introduce the relation between abstract states and concrete states:

Definition 8 (State abstraction). *$Abstract_S(\Sigma_C, \Sigma_A)$ holds if and only if* $\{(ds, \theta) \mid \exists \ r, \Sigma_C(r) = (t, \theta, ds, or, \pi, receivers, bg)\} = Domain(\Sigma_A)$ *and* $\forall \ r, \ \forall \ ds, \ \forall \ \theta, \Sigma_C(r) = (t, \theta, ds, or, \pi, receivers, bg) \iff$ $\exists \ v \mid \Sigma_A(ds, \theta) = (t, or, v, \pi, receivers, bg)$.

The relation $Abstract_L$ denotes that a trace is an abstraction of a log:

Definition 9 (Log abstraction). *$Abstract_L(L, \sigma)$ holds if and only if there exists a function Map such that $Map : \mathbb{N} \rightarrow \mathcal{P}(\mathbb{N}) \mid \forall \ r \in [1, |\sigma|], Map(r) \neq \varnothing \ \wedge \ \forall \ r, s \in [1, |\sigma|], \ \forall \ r' \in Map(i), \ \forall \ s' \in Map(j), \ r < s \Longrightarrow r' < s'$ and for all $i \in [1, |\sigma|]$ and for all $j \in [1, |L|]$, the properties in Fig. 3 are true.*

Using this *Abstract* function, it is now possible to express the core correctness property relating traces and logs:

Property 1 (Correctness).

$$Compliant_C(L) \ \wedge \ Abstract_L(L, \sigma) \Longrightarrow Compliant_A(\sigma)$$

$$\begin{array}{l}
\mathit{Map}(i) = \{j\} \ \wedge\ \sigma_i = (\mathit{Disclosure}, t, \mathit{or}, \mathit{ds}, \theta, v, \pi) \iff \\
L_j = (\mathit{Receive}, \mathit{Disclosure}, t, \mathit{or}, \mathit{ds}, \theta, \pi, \mathit{ref}) \ \wedge \\
\mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \\
\\
\mathit{Map}(i) = \{j\} \ \wedge\ \sigma_i = (\mathit{DeleteReq}, t, \mathit{or}, \mathit{ds}, \theta) \iff \\
L_j = (\mathit{Receive}, \mathit{DeleteReq}, t, \mathit{or}, \mathit{ds}, \theta) \wedge \mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \\
\\
\mathit{Map}(i) = \{j\} \ \wedge\ \sigma_i = (\mathit{AccessReq}, t, \mathit{ds}, \theta) \iff \\
L_j = (\mathit{Receive}, \mathit{AccessReq}, t, \mathit{ds}, \theta) \ \wedge\ \mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \\
\\
\mathit{Map}(i) = J \ \wedge\ \sigma_i = (\mathit{Delete}, t, \mathit{ds}, \theta) \iff \\
\forall\ r \in \mathit{Locations}(L, \mathit{min}(J), \mathit{ds}, \theta),\ \exists\ j \in J \mid \\
L_j = (\mathit{Delete}, t, r) \ \wedge\ \mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \\
\\
\mathit{Map}(i) = \{j\} \ \wedge\ \sigma_i = (\mathit{DeleteOrder}, t, \mathit{tp}, \mathit{ds}, \theta) \iff \\
L_j = (\mathit{Send}, \mathit{DeleteOrder}, t, \mathit{tp}, \mathit{ds}, \theta) \wedge \mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \\
\\
\mathit{Map}(i) = \{j\} \ \wedge\ \sigma_i = (\mathit{Forward}, t, \mathit{rec}, \mathit{ds}, \theta, v, \pi) \iff \\
L_j = (\mathit{Send}, \mathit{Val}, t, \mathit{rec}, \mathit{ref}) \text{ with } \mathit{State}_C(L, j-1)(\mathit{ref}) = \\
(t', \theta, \mathit{ds}, \mathit{or}, \pi, \mathit{receivers}, \mathit{bg}) \ \wedge\ \mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \\
\\
\mathit{Map}(i) = \{j\} \ \wedge\ \sigma_i = (\mathit{Use}, t, \mathit{ds}, \theta, \mathit{purpose}, \mathit{reason}) \ \wedge \\
\mathit{State}_A(\sigma, i-1)(\mathit{Context}) = \mathit{ct} \iff \\
L_j = (\mathit{Read}, t, \mathit{ref}, \mathit{purpose}, \mathit{reason}) \text{ with } \mathit{State}_C(L, j-1)(\mathit{ref}) = \\
(t', \theta, \mathit{ds}, \mathit{or}, \pi, \mathit{receivers}, \mathit{bg}) \ \wedge\ \mathit{Abstract}_S(\mathit{State}_C(L, j-1), \mathit{State}_A(\sigma, i-1)) \ \wedge \\
\mathit{State}_C(L, j-1)(\mathit{Context}) = \mathit{ct}
\end{array}$$

Fig. 3. Log abstraction definition

This property shows that the abstract meaning of the policies (which can be understood by users) reflect the actual properties of the logs. It also makes it possible to abstract the log into a trace and analyse the trace instead of the log.

Proof outline: Since $\mathit{Compliant}_A(\sigma)$ is defined as the conjunction of the seven trace compliance hypotheses Ai defined in §2, it is equivalent to show that they all hold. We do not detail all proofs here but present the strategy and an archetypal example[5]. Generally speaking, starting with the premise of a given Ai, one wants to reach the corresponding conclusion, assuming the ad hoc log compliance property Ci and $\mathit{Abstract}_L(L, \sigma)$. Abstract events can be mapped back to one or more concrete events; for instance, in case of deletion, all references for a given ds and θ must be deleted, giving rise to multiple concrete *Delete* events. The corresponding log compliance property is then used. Often, to use the log compliance property in question, information about states is needed and can be obtained through the state abstraction used in the predicates. For instance, in the case of $A7$, concluding that $\pi.fw \neq \uparrow$ via $C7$ implies reasoning over the

[5] See [6] for more details.

concrete state associated to the reference parameter of the $(Send, Val, \ldots)$ event; indeed, the event itself does not carry the associated policy, unlike its abstract version $Forward$, but the state mapping is realised through $Abstract_L(L, \sigma)$.

The case of $A2$ is typical: its assumptions are $\sigma_i = (Delete, t', ds, \theta) \ \wedge\ State_A(\sigma, i-1)(ds, \theta) = (t, or, v, \pi, receivers, bg)$. We assume $Abstract_L(L, \sigma)$. Let $J = Map(i)$. The part of $Abstract_L(L, \sigma)$ relative to $Delete$ yields $\forall\ r \in Locations(L, min(J), ds, \theta),\ \exists\ j \in J \mid L_j = (Delete, t', r) \wedge Abstract_S(State_C (L, j{-}1), State_A(\sigma, i{-}1))$, Since $State_A(\sigma, i{-}1)(ds, \theta) = (t, or, v, \pi, receivers, bg)$, we get, in particular, $\forall\ r \in Locations(L, min(J), ds, \theta),\ \exists\ j \in J \mid State_C(L, j-1)(r) = (t, \theta, ds, or, \pi, receivers, bg)$. $C2$ can now be used, and gives $\forall\ (t_p, l) \in receivers,\ \exists\ k \mid \exists\ t'' \mid L_k = (Send, DeleteOrder, t'', t_p, ds, \theta)\ \wedge\ k \in]\alpha, i[$ with $\alpha = max\{n \mid (t_p, n) \in receivers\}$. Using $Abstract_L(L, \sigma)$ again for $DeleteOrder$ yields the desired conclusion: $\forall\ (t_p, l) \in receivers,\ \exists\ k' \mid Map(k') = \{k\} \mid \sigma_{k'} = (DeleteOrder, t'', tp, ds, \theta)$ with $k' \in]\alpha, j'[$ and $\alpha = max\{n \mid (t_p, n) \in receivers\}$. In this case, it is critical to establish a correspondence between abstract and concrete states to be able to reason over the $receivers$ set that features in the conclusion of both properties. In the case of $A6$ and $C6$, context equivalence is used.

Race conditions From the perspective of a DS, it is essential that all copies of data are actually deleted in the end, whether they are local or remote. The following property guarantees that all deletion requests are eventually fulfilled on all levels:

Property 2 (Absence of Race Conditions). All deletion requests are fulfilled after a finite delay, provided the log is compliant and of finite length.

Proof outline: We assume $L = L_1 \ldots L_n$ to be a log of length n, ds and θ fixed. All deletion requests are fulfilled after a finite delay. Indeed, assume $\exists\ i \in [1, n] \mid L_i = (Receive, DeleteReq, t, or, ds, \theta),\ L_i \in L$ and $A = Locations(L, i, ds, \theta)$. By contradiction, the following alternatives are impossible:

- Assume there exists a local copy of the initial data which is never deleted, i.e. $\exists\ ref \in A \mid \forall\ s \in [1, n], L_s \neq (Delete, t', ref)\ \wedge\ L_s \neq (Copy, t'', ref', ref)$ with $ref' \notin Locations(L, i, ds, \theta)$ — this contradicts $C3$.
- Assume there is a third party whom the data was shared with and who never received a $DeleteOrder$, i.e. $\exists\ \alpha \in AllReceivers(L, i, ds, \theta)$ and $\forall\ r \in [1, n], L_r \neq (Send, DeleteOrder, t, \alpha, ds, \theta)$. Because of the above, we know $\exists\ k \mid L_k = (Delete, t', ref)$ with $ref \in A$ — this contradicts $C2$.
- Assume the data was received by the DC from a third party TP after its initial versions were deleted locally at time t', i.e. $\exists\ t'' \mid (Receive, Disclosure, t'', TP, ds, \theta, \pi, ref)\ \wedge\ t'' > t'$. This contradicts $C2$'s guarantee the deletion order to TP was sent out before t', since the deletion order makes the data unavailable to TP at time t''.

On the other hand, there is no guarantee that data for a given θ is deleted at the end of a trace if no deletion request exists for it. Indeed, successive disclosures with ever-growing global deletion delays $\pi.dd$ do not contradict $C1$.

5 Accountability Process

The formal framework presented in this paper contributes to the three types of accountability introduced in §1: it can be used to provide precise definitions of privacy policies and to build log analysers to check the compliance of a log with respect to the privacy policies of the data collected by the DC. Actual log files can be parsed and converted by log abstraction to traces that can be mechanically checked as in [4]. In addition, it suggests a number of manual checks and procedural measures required to complement the log analysis and make it fully effective. In practice, as we argued in [5], a true accountability process should impose that these manual checks are carried out by independent auditors.

The additional manual checks suggested by the formal framework fall into two categories:

- *General verifications on the architecture of the system:* the goal of these verifications is to convince the auditor that the log reflects the actual execution of the system. In general it will not be possible to check this property formally because it will be out of the question to build a formal model of an entire system just for the purpose of accountability. However, the formal framework provides clear guidelines about the guarantees that the DC should provide (in informal or semi-formal ways, for example in the form of diagrams and design documentation). Basically, each type of log event leads to specific assumptions which have to be met by the logging tool and demonstrated by the DC: for example any operation involving the receipt, copy or transfer of personal data should be appropriately recorded in the log, each use of personal data should be associated with a precise purpose recorded in the log, etc.
- *Specific verifications depending on the outcome of the log analysis:* the log contains references to pieces of information that may have to be checked by the auditor. For example, the *reason* argument of *Read* events can take the form of a piece of text explaining in more detail the justification for the use of the data[6]. Similarly, the parameters associated with *break-glass* events can be checked to confirm that they provide sufficient justifications for the breach of a privacy property[7].

It should be clear that the objective of an audit in the context of accountability is not to provide a one hundred per cent guarantee that the system is compliant. The general philosophy is that a good accountability process should make it more difficult for DC to breach the rules and also to cover up their misbehaviour. In practice, auditors (or controllers of Data Protection Authorities[8]) do not attempt to check all log entries for all collected data: they rather choose to explore logs

[6] These descriptions can be recorded in a library and provided through specific functions; they are useful to complement and define more precisely the *purpose* argument.

[7] Each *break-glass* event is associated with a set *et* of affected entities and data types.

[8] Such as the CNIL in France.

selectively to check specific types of data[9]. In our model, the correctness property of §4 defines a condition to be met by such a log analyser. Despite the fact that a full application of formal verifications is out of reach in this context, we believe that the formal approach followed here can bring significant benefits in terms of rigour in the definition of the objectives and the procedures to reach them.

6 Related Work

Accountability in computer science is generally associated with very specific properties. An example of a formal property attached to accountability is non-repudiation: Bella and Paulson [2] see accountability as a proof that a participant took part in a security protocol and performed certain actions. The proof of non-repudiation relies on the presence of specific messages in network history.

Several frameworks for a posteriori compliance control have already been developed. Etalle and Winsborough [11] present a logical framework for using logs to verify that actions taken by the system are authorized. Cederquist et al. [7] introduce a framework to control compliance of document policies where users may be audited and asked to justify actions. Jagadeesan et al. [15] define accountability as a set of mechanisms based on "after-the-fact verification" by auditors for distributed systems. As in [19], blame assignment based on evidence plays a central role in this framework. Integrity (the consistency of data) and authentication (the proof of an actor's identity) are integral to the communication model. Together with non-repudiation [2], these technical concepts are often seen as pillars of the concept of accountability in computer science literature.

On the practical side, Haeberlen [14] outlines the challenges and building blocks for accountable cloud computing. Accountability is seen as desirable both for customers of cloud services and service providers. The building blocks of accountability are defined as completeness, accuracy and verifiability. Technical solutions to enable these characteristics on cloud computing platforms have been devised by the authors.

Work presented in [17] proposes criteria for acceptable log architecture depending on system features and potential claims between the parties.

Finally, current legal perspectives on accountability are surveyed in [13].

7 Conclusions

Considering the ever-growing collection and flow of personal data in our digital societies, a priori controls will be less and less effective for many reasons, and accountability will become more and more necessary to counterbalance this loss of *ex ante* control by DS. Another major benefit of accountability is that it can act as an incentive for DC to take privacy commitments more seriously and put appropriate measures in place, especially if audits are conducted in a truly independent way and possibly followed by sanctions in case of breach. As pointed

[9] Typically, sensitive data or data for which they have suspicions of breach.

out by De Hert, “the qualitative dimension of accountability schemes may not be underrated” [10].

However, the term “accountability” has been used with different meanings by different communities, very often in a broad sense by lawyers and in very specific technical contexts by computer scientists. This paper aims to reconcile both worlds, by defining precisely the aspects which can be formalised and showing how manual checks can complement automatic verifications.

The language used here to express privacy policies and the sets of events are typical of the most relevant issues in this area, but they should obviously be complemented to be used as a basis for an effective accountability framework. In order to implement such a framework, several issues should be addressed:

- The security (integrity and confidentiality) of the logs should be ensured. This aspect, which has not been discussed here, has been addressed by previous work [3, 20, 21].
- A suitable interface should be provided to the auditors for a selective search of the logs based on an analyser meeting the requirements defined in §4. This interface must provide convenient ways for the auditor to reach the documents that need complementary verifications.
- More complex data manipulation operations should be considered, including for example the merging of different pieces of personal data or anonymization techniques. The privacy policy language should be extended to allow the DS to specify the rules associated with the result of such operations.

Last but not least, it is also possible to reduce even further the amount of data stored in the logs by ensuring that not only the values of personal information are not recorded in the logs, but also the identity of the DS and the type of data (the (ds, θ) pair in the formal model). Indeed, the only role of this pair in the model is to establish a link with the privacy policy and it could as well be anonymized through a hash function. The fact that our formal model can be used to implement an effective accountability framework without recording any extra personal data makes it possible to counter the most common objection against accountability in the context of personal data protection. This argument is especially critical for Data Protection Agencies, for which such a “personal-data-free” accountability framework could significantly ease day-to-day checks. It can also be a key argument for DC reluctant to create new logs which may represent additional security risks. For these reasons, we hope this work can pave the way for future wider adoption of effective accountability of practice.

Acknowledgement This work was partially funded by the European project PARIS / FP7-SEC-2012-1 and the Inria Project Lab CAPPRIS (Collaborative Action on the Protection of Privacy Rights in the Information Society).

References

1. Article 29 Data Protection Working Party: Opinion 3/2010 on the principle of accountability (2010)

2. Bella, G., Paulson, L.C.: Accountability Protocols: Formalized and Verified. ACM Trans. Inf. Syst. Secur. 9(2), 138–161 (2006)
3. Bellare, M., Yee, B.S.: Forward Integrity for Secure Audit Logs. Tech. rep., University of California at San Diego (1997)
4. Butin, D., Chicote, M., Le Métayer, D.: Log Design for Accountability. In: 2013 IEEE Security & Privacy Workshop on Data Usage Management. pp. 1–7. IEEE Computer Society (2013)
5. Butin, D., Chicote, M., Le Métayer, D.: Strong Accountability: Beyond Vague Promises. In: Gutwirth, S., Leenes, R., De Hert, P. (eds.) Reloading Data Protection, pp. 343–369. Springer (2014)
6. Butin, D., Le Métayer, D.: Log Analysis for Data Protection Accountability (Extended Version). Tech. rep., Inria (2013)
7. Cederquist, J., Corin, R., Dekker, M., Etalle, S., den Hartog, J., Lenzini, G.: Audit-based compliance control. Int. J. Inf. Secur. 6(2), 133–151 (2007)
8. Center for Information Policy Leadership: Data Protection Accountability: The Essential Elements (2009)
9. Colin J. Bennett: Implementing Privacy Codes of Practice. Canadian Standards Association (1995)
10. De Hert, P.: Accountability and System Responsibility: New Concepts in Data Protection Law and Human Rights Law. In: Managing Privacy through Accountability (2012)
11. Etalle, S., Winsborough, W.H.: A Posteriori Compliance Control. In: Proceedings of the 12th ACM symposium on Access control models and technologies. pp. 11–20. SACMAT '07, ACM (2007)
12. European Commission: Proposal for a Regulation of the European Parliament and of the Council on the Protection of Individuals with Regard to the Processing of Personal Data and on the Free Movement of such Data (2012)
13. Guagnin, D., Hempel, L., Ilten, C.: Managing Privacy Through Accountability. Palgrave Macmillan (2012)
14. Haeberlen, A.: A Case for the Accountable Cloud. Operating Systems Review 44(2), 52–57 (2010)
15. Jagadeesan, R., Jeffrey, A., Pitcher, C., Riely, J.: Towards a Theory of Accountability and Audit. In: Proceedings of the 14th European conference on Research in computer security. pp. 152–167. ESORICS'09, Springer (2009)
16. Joint NEMA/COCIR/JIRA Security and Privacy Committee (SPC): Break-Glass: An Approach to Granting Emergency Access to Healthcare Systems (2004)
17. Le Métayer, D., Mazza, E., Potet, M.L.: Designing Log Architectures for Legal Evidence. In: Proceedings of the 8th international conference on Software engineering and formal methods. pp. 156–165. SEFM'10, IEEE Computer Society (2010)
18. Organisation for Economic Co-operation and Development: OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data (1980)
19. Schneider, F.B.: Accountability for Perfection. IEEE Security & Privacy 7(2), 3–4 (2009)
20. Schneier, B., Kelsey, J.: Secure Audit Logs to Support Computer Forensics. ACM Trans. Inf. Syst. Secur. 2(2), 159–176 (1999)
21. Waters, B.R., Balfanz, D., Durfee, G., Smetters, D.K.: Building an Encrypted and Searchable Audit Log. In: Proceedings of the Network and Distributed System Security Symposium. NDSS 2004 (2004)