The Timed Decentralised Label Model

Pedersen, Martin Leth; Sørensen, Michael Hedegaard; Lux, Daniel; Nyman, Ulrik Mathias; Hansen, Rene Rydhof

Published in:
Secure IT Systems

DOI (link to publication from Publisher):
10.1007/978-3-319-26502-5_3

Publication date:
2015

Document Version
Peer reviewed version

The Timed Decentralised Label Model

Martin Leth Pedersen[1], Michael Hedegaard Sørensen[1], Daniel Lux[2], Ulrik Nyman[1], and René Rydhof Hansen[1]

[1] Department of Computer Science, Aalborg University, Denmark
{mhso10,mped10}@student.aau.dk, {ulrik,rrh}@cs.aau.dk
[2] Seluxit, Aalborg, Denmark
daniel@seluxit.com

1 Introduction

By some estimates, the number of devices connected through the so-called *Internet of Things* (IoT) will reach the 50 billion mark in 2020[3]. While forecasting such numbers is not an exact science, it seems clear that in the near future, a very large number of Internet connected devices will be deployed everywhere, not least in our homes, e.g., in the form of smart meters, refrigerators, and other household appliances, facilitating the "smart home" of the future. However, filling our homes with sensors and devices able to measure, monitor, and report on all activities, immediately raises questions about how security and privacy can be handled satisfactorily.

Of particular importance for security is the fundamental question of how to model the security (and privacy) policies such a system must comply with: the highly distributed and decentralised nature of the underlying system does not fit well with the classic policy models, such as Bell-LaPadula [5] or Clark-Wilson [8], nor with traditional MAC/DAC access control models. This is further complicated by the fact that security and privacy policies in IoT systems must be able to cope with *time dependent* elements, e.g., a smart meter may only send (aggregate) measurements every 15 minutes in order to preserve privacy. None of the classical models mentioned above incorporate timing constraints and although a number of temporal aspects, in various security models, are discussed in [17], focus is *information release* and they only indirectly deal with real-time constraints on security policies. One possible solution would be to encode temporal constraints/policies into the highly flexible formalism of *Flow Locks* [7, 6]. However, based on our experience in modelling, analysing, and verifying the safety of systems that incorporate real-time timing constraints, we have found that safety- and security-properties involving real-time clocks and constraints are often subtle and counter-intuitive. We therefore believe it is important that time is represented *explicitly* in security models for IoT.

In this paper we propose the *Timed Decentralised Label Model* (TDLM) as a step towards a modelling formalism for IoT security policies and illustrate this

[3] http://www.brookings.edu/blogs/techtank/posts/2015/06/9-future-of-iot-part-2

for a non-trivial smart meter system. As the name suggests, the TDLM is an extension of the decentralised label model (DLM) with a time component [14, 13], formalised here by *timed automata* [1]. Given the decentralised structure of typical IoT systems, as noted above, the DLM seems like a natural match with its emphasis on local (decentralised) control; similarly timed automata have a long history of modelling systems with time components. It bears mentioning, that instead of the DLM, we could instead have extended the Flow Locks formalism mentioned above with timed automata, potentially yielding a more flexible approach. We leave this for future work.

To illustrate the features of the TDLM, we have chosen a *smart meter* use case. Smart meters were chosen because the relevant security policies are quite complex with a non-trivial time component. Furthermore, smart meters are being introduced on a massive scale throughout the European Union and are expected to replace "dumb" meters in the coming years. A smart meter is a device responsible for monitoring power consumption in households and reporting this to electrical companies in order to ensure correct billing and to enable "smart" use of power, e.g., by (automatically) postponing certain power consuming tasks to a time of day when prices are low, e.g., doing the laundry at night. There are many security and privacy aspects to take into account when designing and implementing a smart meter system: a major privacy concern, which is also the main focus of this paper, is that power companies can build highly detailed power consumption profiles for individual homes. To prevent this, power companies are only allowed to access the measured power consumption at certain time intervals [11]. For further smart meter security issues, see [3, 2].

The work reported on in this paper, is a simplified and summarised version of the work done in the master's thesis of the two first authors [16].

2 Preliminaries

In this section we briefly review the main theories underlying the work in this paper: the decentralised label model (DLM) and timed automata.

2.1 The Decentralised Label Model

The fundamental idea of the DLM, is that every *principal* that contributes information to a system, should be allowed to define a security policy for how the contributed data can be used (in that system) [15]. Principals are the authority entities, or actors, in a system, e.g., users, groups, or roles, between which data can flow (through data channels).

In order to capture common access control idioms, such as individuals acting for a group or vice versa, principals in the DLM are ordered into a hierarchy, the so-called *principal hierarchy*, through the *act-for* relation. A principal a allowed to *act-for* another principal b, denoted $a \succeq b$, intuitively inherits all the privileges of b, i.e., can access the same data as b. In practice, the *act-for* relation can be specialised to more specific privilieges, e.g., reading a specific file. The *act-for*

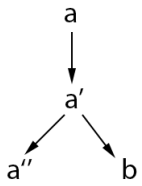

Fig. 1. Example principal hierarchy: $a \succeq a'$, $a' \succeq a''$, and $a' \succeq b$.

relation is taken to be reflexive and transitive; an example principal hierarchy is shown in Figure 1.

In the DLM, data (or rather data sources, sinks, and channels) can be annotated with *labels* expressing the (security) policies that should hold for that data. A label is composed of a set of *security policies*, where each security policy comprises an *owner* and a corresponding set of *readers* the owner wishes to permit access to the labelled data. Formally:

$$\mathsf{Label} = \mathcal{P}(\mathsf{SecPol}) \qquad \mathsf{SecPol} = \mathsf{Owner} \times \mathcal{P}(\mathsf{Reader})$$

In keeping with tradition, we write $o\colon R$ for $(o, R) \in \mathsf{SecPol}$. Intuitively, only readers that are permitted by all owners (of a particular label) are permitted to access data with that label. This set of readers is called the *effective set of readers* and is formalised as follows (for $L \in \mathsf{Label}$)

$$\mathit{effectiveReaders}(L) = \bigcap_{o \in \mathit{owners}(L)} \mathit{readers}_{\succeq}(\mathit{readers}(L, o))$$

where $\mathit{owners}(L) = \{o \mid (o, R) \in L\}$, $\mathit{readers}(L, o) = \{r \mid (o, R) \in L, r \in R\}$, and $\mathit{readers}_{\succeq}(R) = \{p \mid \exists p' \in R\colon p \succeq p'\}$. A policy with no owner and an empty reader set is equivalent to allowing all principals in the system access [15].

In order to facilitate and reason about the flow of labelled data through a system with varying labels, data can be *relabelled* either by *restriction* or *declassification*. The latter is an intended and deliberate "leak" of data, while the former allows data to flow to actors with a more strict security policy. A relabelling from a label L_1 to another label L_2 is called *safe*, denoted $L_1 \sqsubseteq L_2$ if the relabelling is a *restriction*, i.e., if L_2 is at least as restrictive as L_1, intuitively that means L_2 has fewer readers and/or more owners. Taking the principal hierarchy into account, we can formalise safe relabelling as follows

Definition 1 (Safe Relabelling). *Let* $L_1, L_2 \in \mathsf{Label}$ *and* $I_1, I_2 \in \mathsf{SecPol}$ *and define* safe relabelling*:*

$$L_1 \sqsubseteq L_2 \equiv \forall I_1 \in L_1\colon \exists I_2 \in L_2\colon I_1 \sqsubseteq I_2$$

where $I_1 \sqsubseteq I_2$ *iff* $o_2 \succeq o_1$ *and* $\mathit{readers}_{\succeq}(R_2) \subseteq \mathit{readers}_{\succeq}(R_1)$ *for* $I_1 = (o_1, R_1)$ *and* $I_2 = (o_2, R_2)$*.*

The above definition corresponds to the complete relabelling rule in [15]. For a proof that the complete relabeling rule is both sound and complete with respect to the formal semantics see [12]; meaning that the rule only allows safe relabelings and allows all safe relabelings.

When executing a program within a system, values are often derived from other values, for example a new value may be derived by multiplying two other values. In the DLM a derived value v must have a label that enforces the policies of the values used to derive v meaning that the label of v must be at least as restrictive as the combined label of the operands. Formally if we have two operands labeled with the labels L_1 and L_2 respectively the label for a derived value would be a join of these two which in terms is the union of the labels joined, as described in Definition 2.

Definition 2 (Label Join). *Let* $L_1, L_2, \in \mathsf{Label}$ *define* label join*:*

$$L_1 \sqcup L_2 = L_1 \cup L_2$$

Intuitively, the label join produces the *least* label that restricts both L_1 and L_2.

The security policies mentioned so far have all been *confidentiality* policies that only considers who can observe data. However, the DLM can also specify *integrity* policies which considers who are able to modify the data protected by the policies. The integrity policies are quality guarantees provided by the owners of the policies that only the specified writers have modified the data. The syntactical notation is similar to the notation for confidentiality policies but instead of a reader set, a writer set is associated with the policies. An integrity policy can also be relabelled in a manner similar to that of a confidentiality policy [15]. We do not go into further details with integrity policies or the DLM here, but refer instead to [12, 15].

2.2 Timed Automata

Finite automata are well known theoretical computation models used for describing logical program behavior based on transitions and states that a program can be in. An extension of the finite automata formalism with time is called a *timed automaton*, which adds a finite set of real-valued clocks to the finite automaton [1]. The clocks are increased at the same rate and can be reset during a transition if need be. The primary use of the clocks is to set up guards that can prevent transitions from being executed or prevent the program from being in a certain state.

A timed automaton is a tuple $T = (\Sigma, L, L_0, C, E)$ consisting of the following components:

- Σ is the input alphabet accepted by the automaton.
- L is the set of possible finite locations that the automaton can be in.
- L_0 is the set of start locations which is a subset of L; $L_0 \subseteq L$
- C is a finite set of clocks

– E is the set of possible transitions in the automaton, formally defined as $E \subseteq L \times L \times [\Sigma \cup \{\epsilon\}] \times 2^C \times \Phi(C)$

An edge in the timed automaton is then defined as $\langle s, s', \sigma, \lambda, \delta \rangle$, which represents a transition from a program location s to another program location s' on the input σ. λ is then the set of clocks that will be reset with the transition, and δ is the enabling condition (the guard). The automaton starts in one of the start locations with all clocks set to zero. The clocks then increase to reflect time elapsed and a transition may be taken when the guards of an edge is satisfied.

An example of a system that is ideally modelled with a timed automaton is a simple smart meter system consisting of a smart meter and a power company. The power company must be restricted to only being able to read the smart meter data at certain time intervals. In Figure 2 this scenario is presented with a timed automaton constructed in the Uppaal model checker [4][4]. Uppaal is used to verify timed automata models, such as the model presented in Figure 2. The automaton consists of two locations *smd* and *ec*, where *smd* is the smart meter data and *ec* is the power company. The system can take the transition from *ec* to *ec* non-deterministically while waiting for the clock x to be larger than 90 (days). When the clock x is larger than or equal to 90, the transition guarded by the expression $x >= 90$ can be taken thus modelling a read of smart meter data by going to the location *smd*, where the only transition that can be taken resets the clock variable to zero and leads back to *ec*. This simple example models an electrical company that can read smart meter data of a single customer every 90 days.

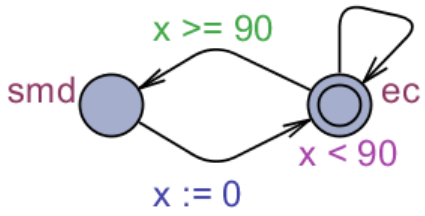

Fig. 2. Simple smart meter example where an electrical company reads smart meter data every 90 days. The location with the double circle is a start location.

However, the simple model also reveals some of the non-trivial issues encountered when formalising access control policies with a time element, e.g., for how long time is access allowed (when can/does the timed automaton leave state *smd*). In a later section, we show how the Uppaal model checker can be used to answer such questions and validate the model.

[4] `http://uppaal.org`

3 The Timed Decentralised Label Model

In the following we define and describe the *timed decentralised label model*. The main idea is to extend DLM policies with *time constraints* formulated over *clock variables* associated with underlying timed automata (see Section 2.2). In this work we assume that clocks used by the system are controlled by the system; this is similar to the assumption in "normal" DLM that the principal hierarchy is under control by the system.

Note that this sidesteps the well-known thorny issue of how to synchronise (real-time) clocks in a distributed system. In fact the TDLM, as explained later, does not necessarily require clocks to be synchronised, but can in fact be used to model systems with several (local) clocks. We leave it for future work to investigate the consequences of this modelling assumption.

We start by discussing the new label constructs added to DLM and follow that by a formalisation and semantics of the new constructs in terms of a network of timed automata.

3.1 TDLM Constructs

To clarify the constructs TDLM adds to the DLM, an explanation of the individual constructs is presented to give an intuitive understanding of the security policies that can be expressed with these. We start with an example of of a (timed) security policy, illustrating all the new constructs by extending a basic DLM security '$o\colon r$':

$$o(x[15;?event;1] > 10 \,\&\&\, y < 5)\colon r[!event]$$

Informally, the above (timed) security policy states that the policy owner, o, allows any stated readers (here only r) access to the labelled data whenever the clock x has a value greater than 10 and the clock y has a value less than 5. Furthermore, the clock x will be reset to 1, whenever it reaches the value of 15 or an event, named '*event*', is triggered. Finally, whenever the reader r accesses the data, an event named '*event*' is triggered (thus resetting the x clock). Note that, in order to be able to reason about time, e.g., how long is there between any two reads of a variable, we have introduced events. This is a departure from other formalisations of secure information flow where the exact time of a read is implicit (or rather: irrelevant). In future work, we will investigate if it is possible to move towards the more traditional approach.

We now discuss the individual constructs in more detail:

- **Declaring clock variables.** Clock variables are used to restrict access to data based on time (x and y in the above example) and are declared within a set of parentheses which is placed after a principal (owner or reader) to restrict that principal's access. If a clock is placed on an owner of a policy then all readers associated with this owner are restricted by the clock, however if a clock is placed on a reader then only that specific reader is restricted

by it. A clock variable is identified by its unique name which can be any combination of alphabetic characters and the value of a clock variable can be any positive integer.

- **Comparing clock values.** Clock variables can be compared to other clock variables and constant integers with the use of the usual binary comparison operators. If that comparison evaluates to true then the clock variables allows access to the entity it is associated with. The comparison is defined within the parenthesis along with the declaration of the clock variable for example $(x > 10)$. The value the clock variable is compared to is called the comparison value.
- **Multiple comparisons.** Multiple clock comparisons can be performed within the same parenthesis by separating them with the logical && and || operators, which evaluates as expected for example $(x > 10 \;\&\&\; x < 15)$ would evaluate to true when x is between ten and 15. A statement of clock comparisons placed within a single set of parenthesis is called a clock expression and must evaluate to true before the associated principal(s) can gain access to the data.
- **Parameterised clock variables.** Clock variables can be parameterised with the use of square parentheses placed after the name of the clock variable. However, if no parameters are defined on a clock variable then the square parentheses may be omitted. Within the square parenthesis three optional parameters can be declared. Parameters are separated with a semicolon and are identified in the order: upper limit, event, and reset value. If only one parameter is declared and that parameter's identifier starts with a question mark then it is an event otherwise it would be an upper limit. A parameterized clock variable would then be defined as $(x[15; ?event; 1] > 10)$.
- **Upper limit.** An upper limit is a constant used to define when a clock variable should be reset. Upon reaching the upper limit the value of the clock variable will instantly get reset.
- **Reset value.** The reset value is a constant used to define the value of a clock variable when it resets. If omitted from a clock variable the reset value is always zero.
- **Events.** Events are defined by a unique alphabetic name starting with a question mark and can be placed as a parameter on a clock variable, which indicates that when the event is triggered then the value of the clock variable is reset.
- **Event trigger.** An event trigger can be specified on any principal, and is placed immediately after a principal's identifier. An event trigger starts with the symbol ! followed by the name of the event to be triggered when the corresponding principal successfully reads data. Several event triggers can be placed on the same principal in the same security policy by separating the event triggers with a comma, for example $p[!event1, !event2]$ would trigger *event1* and *event2* when p reads the data. An event can only be triggered from within the system by those principals that have an event trigger defined on them in a given security policy.

We now proceed to the formal definition of timed security policies and timed labels.

3.2 Formal Definition

In the following, we assume without further specification, the existence of countably infinite sets of *clock variables* $\mathsf{ClockVar}$ and events (or rather event names) Event. Clock declarations are then defined to have the following form:

$$\Upsilon ::= c \mid c[\alpha;?\beta;\gamma]$$

where $c \in \mathsf{ClockVar}, \alpha, \gamma \in \mathsf{ClockVal}$, and $\beta \in \mathsf{Event}$. As discussed above, we allow clock declarations to omit any and all of α, β, and γ in the above, indicating that default values should be used (for α and γ) or that no reset events are defined (for β). We can now define *clock expressions* as combinations of clock declarations. A clock expression is of the form:

$$\Phi ::= \Upsilon \mid \Upsilon \bowtie \Upsilon \mid \Phi \,\&\&\, \Phi \mid \Phi \,||\, \Phi$$

with $\bowtie \in \{<, \leq, ==, !=, \geq, >\}$ representing the standard comparators. As mentioned above, clock expressions Φ can be placed on any principal in a security policy.

Finally, by extending DLM security policies and labels with clock expressions, we are able to give the formal definition of TDLM security policies and labels (again using $\perp$ to denote optional values):

$$\mathsf{SecPol}_T = \mathsf{Owner} \times \Phi_\perp \times \mathsf{Event}_\perp \times \mathcal{P}(\mathsf{Reader} \times \Phi_\perp \times \mathsf{Event}_\perp)$$

Which leads to the following obvious definition of TDLM labels:

$$\mathsf{Label}_T = \mathcal{P}(\mathsf{SecPol}_T)$$

Having defined these, we next turn to the semantics of TDLM labels (and security policies).

3.3 From Policies to Timed Automata

In the following we define the semantics of TDLM labels and security policies by translating them into a network of timed automata. In essence, TDLM security policies describe which principals can access protected data and when.

The behavior of security policies can be expressed via one or more timed automata where each principal present in the security policy is associated to a timed automaton describing their access restrictions. However if several principals are allowed to access the data under the same conditions then a single timed automaton might describe the access restrictions for multiple principals. A timed automaton that describes the access restriction for a principal has its start location labeled with the name of the principal that the automaton describes as

depicted in Figure 3, where rw_i is the principal that the timed automaton depicts the access possibility of. The location named *data* is used to describe that when the timed automaton is in this location, the principal is allowed to observe the data. Note however, that the location is *committed* (denoted with a C) indicating that an edge going away from this location *must be taken immediately* to enforce the access restriction. The access restriction must be placed on an ingoing edge to the *data* location thus modeling that the data is protected by some time constraints Φ. If a policy contains events β that should be triggered by certain principals then the timed automaton that describes the access behavior of these principals must trigger the event on the edge going away from the *data* location thus modeling reset only on successful read/write.

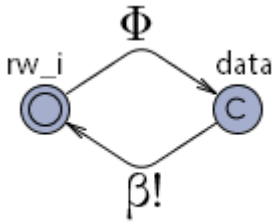

Fig. 3. General timed automaton that describes the access restrictions for the principal rw_i restricted by the clock expression Φ and triggering the event β on successful read/writes.

In addition to this, for each clock variable in a security policy, a timed automaton is used to describe how this clock variable is incremented and who can increment it. A timed automaton that describes a clock variable consists of one location with up to three cyclic edges that describe incrementing the value of the clock c, resetting the clock value upon reaching an upper limit α, and resetting the clock value when an event β is triggered. Note that we use UPPAAL channels as a natural fit for modelling for events. The clock value is reset to the reset value γ when an event is triggered or an upper limit is reached. When an upper limit is present on a clock variable then a guard on the incrementation edge must be placed to force a reset when reaching the upper limit. Figure 4 depicts a timed automaton that describes the behavior of the clock c which can be incremented by the principal *co*.

As an example consider the following TDLM security policy:

$$\{o\,(x[20; ?reset; 5] > 10 \;\&\&\; y > 15) : r[!reset]\} \tag{1}$$

The timed automata in Figure 5 formalises this policy. The first timed automaton labelled 1. describes the access restrictions for the owner o which is restricted by the guard placed on the edge from the initial location to the *data* location meaning that the owner o may only observe the data when the conditions for

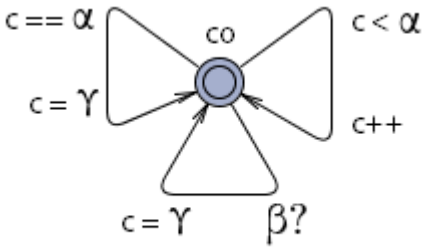

Fig. 4. General timed automaton that describes a clock c that can be incremented by the principal *co*.

x and y are met. When o has successfully gained access to the data (the timed automaton is in the location named *data*) the timed automaton immediately goes to the initial location such that the restrictions placed on the data can be enforced correctly.

Since o and r are restricted by the same clocks they should be modelled by the same timed automata, however r triggers an event when successfully reading the data and thus the access behavior for r must be modelled by a different timed automaton. The timed automaton labeled 2. models the access behavior of r and is equivalent to the timed automaton modeling o except for the event ($reset!$) that is triggered when r has successfully gained access to the data. Events are expressed by channel synchronization meaning that when a timed automaton takes an edge marked with an event trigger such as $reset!$ then all[5] the corresponding timed automata in the system take any available matching edges marked with the same event name such as $reset?$.

The timed automaton labeled 3. models the behavior of the clock variable x, which can be incremented by the principal o (indicated by the name of the start location) when the value of x is below 20 but upon reaching the value 20, o must reset the clock to five before the clock can be incremented again thus modeling the upper limit and reset value of x described by the policy. The edge marked with $reset?$ models the event that is triggered by r and resets the clock to its reset value when the event is triggered.

The timed automaton labeled 4. models the behavior of the clock variable y which also can be incremented by the principal o but this timed automaton contains only one edge as there is no upper limit, reset value or event associated with y.

[5] This is the case as we use UPPAAL's *broadcast channels* in these models

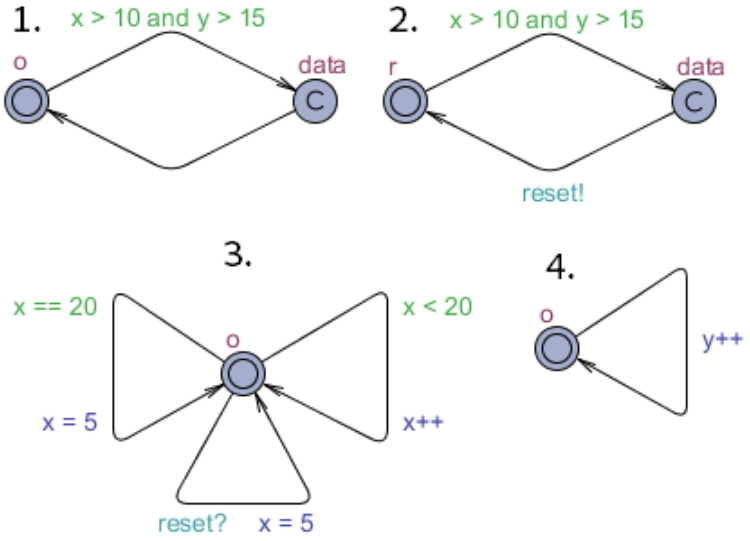

Fig. 5. Timed automata describing the security policy $\{o(x[20; ?reset; 5] > 10 \,\&\&\, y > 15) : r[!reset]\}$

4 Smart Meters: A Case Study

To explore and illustrate the capabilities and expressiveness of the TDLM, a case study of a real world scenario is presented here in some detail. The case study involves a smart meter system which consists of multiple users (household owners and residents), smart meters, power companies, distribution companies, smart meter manufactures, third parties, a data hub and a government entity as depicted in Figure 6. A definition of each entity in the smart meter system is as follows:

- **Users.** Two types of users exist in the smart meter system - household owners and residents. Household owners own one or more households, which each have a smart meter installed, and may also be residents in the households they own. Residents are permitted to live in a household by the household owner.
- **Smart meters.** A smart meter is responsible for collecting electricity consumption data for the household it is installed in. Furthermore, it also serves as a platform for other devices to connect to and communicate with for example for doing home automation.
- **Power companies.** A power company is responsible for delivering electricity to one or more customers and billing the customers according to their electricity consumption.
- **Distribution companies.** A distribution company is responsible for maintaining the power distribution grid and keeping track of which electrical

companies users are associated with. Furthermore, it is responsible for processing raw smart meter data and making this data available to electrical companies via a data hub.

- **Data hub.** The data hub serves as a storage center for processed smart meter data which electrical companies or third parties can gain access to when appropriate.
- **Smart meter manufacturers.** Smart meter manufacturers are responsible for producing the smart meters and updating the firmware if need be.
- **Third parties.** Third parties are entities that might have an interest in the data collected by the smart meters such as research companies.
- **Government.** The government is interested in obtaining power consumption reports such that they can optimize the smart grid.

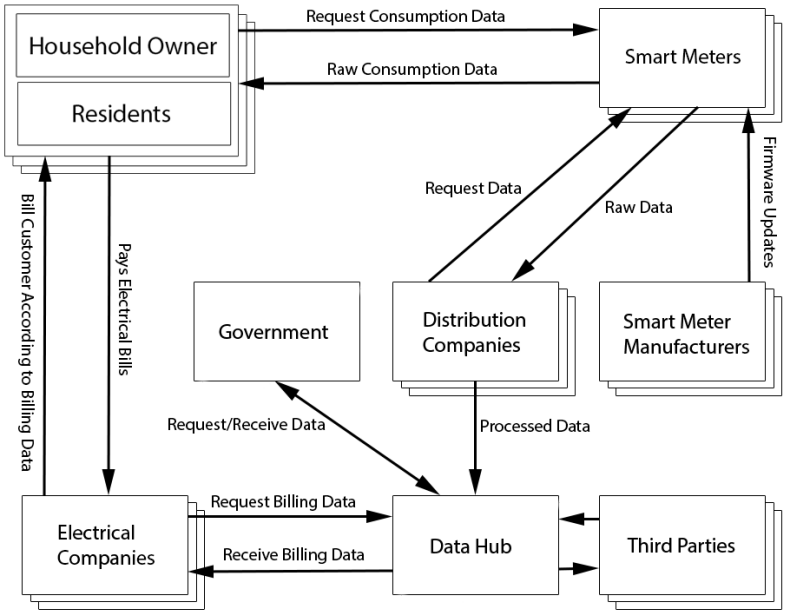

Fig. 6. Overview of the smart meter system with communication channels outlined.

The users can monitor their own power consumption by directly communicating with the smart meter in their household, in order to identify how power can be saved. However, the household owner is capable of granting and revoking access to the smart meter, e.g., for residents living in the household. The distribution companies request data from the smart meters they are associated with and perform necessary processing of the data in order to protect the privacy of

the users. The processed data is delivered to a data hub which enforces which entities that can gain access to the data, when they can gain access to the data, for how long they may access the data, and the granularity of the data that may be accessed. The government and potential third parties can access the data they need through the data hub, however the permissions of each entity may vary for example a research company may be able to access more fine-grained data than the government. The power companies deliver electricity to the users and bill them according to billing data obtained from the data hub and as such an electrical company is implicitly associated with one or more smart meters/users. Note that power companies are not able (or allowed) to access the detailed usage statistics for a single household; depending on the particular setup, such information may or may not be available to distribution companies.

4.1 Smart Meter Privacy Concerns and Access Rights

A smart meter system makes it possible to observe fine-grained electricity consumption data of users associated with a smart meter. The system allows users to observe their own power consumption down to the minute which opens up the possibility of optimizing their usage patterns to lower overall power consumption [11]. In addition to this, the smart meter system provides users with a platform for home automation which can further lower their power consumption by automatically turning off unused devices or starting devices based on electricity prices [3]. However, the possibility of reading fine-grained power consumption data also posses several privacy issues due to the possibility of deriving personal behavior patterns from the power consumption data, for example it is possible to figure out if a person is home during their sick leave or if they left late for work by observing when they consume power [11]. As such it would be favorable to regulate whom that has access to this data and in how fine a granularity the data can be extracted in relation to the entity that wants to observe the data.

As there are several privacy and security issues in regards to smart meters, it should *not* be possible for all entities in- and outside the system to obtain data collected by smart meters or data transferred internally between entities. The smart meter collects data about the user's electrical consumption and as such the data collected by the smart meter are directly related to the user, meaning that there is no need to restrict the user's access to the data in any form. The user is able to observe and analyze the data collected by the smart meter at any time via for example a web interface.

4.2 Smart Meter System Modelled with the TDLM

The smart meter system cannot be described by the DLM alone as this model lacks the possibility of defining time-based security policies, which are crucial in regards to modeling smart meter security. However, the TDLM extends the DLM with the required components for describing a smart meter system in regards to secure information flow and access control.

The principal hierarchy in the smart meter system consists of all the users (u), the smart meters (s) associated with those users, the power companies (e) associated with users, distribution companies (d) associated with smart meters, the data hub (dh), smart meter manufactures (m) and the government (g). The acts-for relationship between these principals is then given as $d \succeq s$ and $s \succeq u$.

The smart meter data can be divided into three segments; the first containing the personal information about the user that is associated with the smart meter at the current time, the second containing the real-time electrical readings recorded by the smart meter, and the third being the smart meter firmware which controls how the smart meter behaves. The first part of the data is owned by the user as it is sensitive information about them, but the electrical company that is associated with the user at the current time is allowed to read the data for billing purposes: $\{u_i : e_j\}$. In addition to this, the data should have an integrity policy which expresses that the user is the owner of the data but trusts the electrical company to change the data if needed. However, in this paper we are only concerned with confidentiality properties and refer to [16] for the full case study.

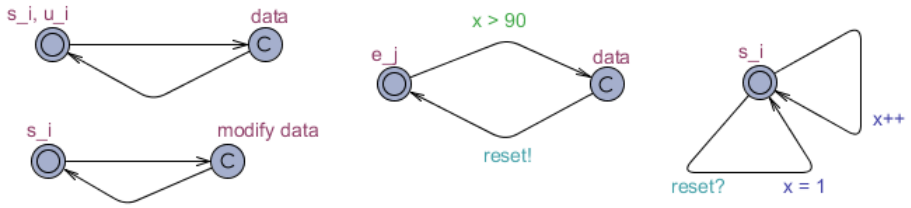

Fig. 7. The smart meter security policies modelled with four timed automata.

The second part of the smart meter data comprises the actual power consumption readings saved by the smart meter every minute. Since this data could reveal a user's private behaviourl pattern, it should only be accessible to the power company in aggregate form. To enforce this, the TDLM introduces time-based security policies formalising that the power company only should have access to the data once in a while, e.g., every quarter, which is the requirement for quarterly billing. Formally, this is captured by the following TDLM policy:

$$\{s_i : u_i, e_j(x[?reset : 1] > 90)[!reset]\} \tag{2}$$

The semantics of this security label is depicted by the four timed automata shown in Figure 7.

Figure 8 gives an overview of the TDLM annotations needed for the entire system. For lack of space we are unable to go into further details with the case study here, but refer to [16] for the full case study. From the above discussion, it

seems that the TDLM is a good fit for modelling many of the security properties relevant for smart meters, in particular those involving timing elements.

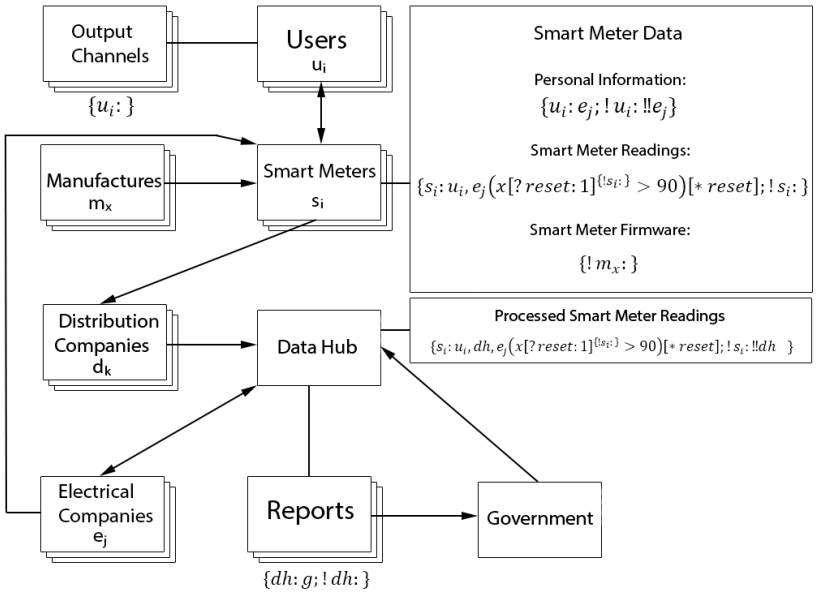

Fig. 8. Smart meter system with security labels as per the definition of TDLM. Arrows depict information flow.

5 Automated Verification of Security Goals

A problem common to all security policy formalism is that once the policies reaches a certain level of complexity, it becomes very difficult, if not impossible, to verify that the security policies actually capture the relevant security properties. Since the TDLM is rooted in timed automata, it is possible to leverage the strong tool support for verifying properties of models using timed automata, mainly through [4] model checking. In the remainder of this section, we briefly outline how the UPPAAL model checker can be used to verify properties of a (simple) TDLM policy.

As an example, we take a security goal that is similar to (a simplified version of) the security goals for the smart meter: we wish to verify that certain data can only be read every 15 time units. The security policy enforced by our system is formulated as follows:

$$\{c : reader(x[15] >= 15)\}$$

Clearly, in this case it is trivial to verify manually that the policy satisfies the goal, but in a real system with even a moderate number of non-trivial policies, it quickly becomes infeasible. Instead we turn to the UPPAAL model checker to analyse the so-called *window of opportunity* for reading the labelled data, e.g., by an attacker or an insider. Figure 9 shows a partial screenshot of performing

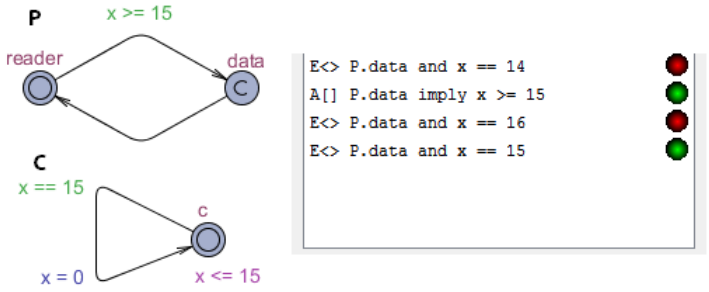

Fig. 9. UPPAAL window of opportunity analysis where upper limit is set to 15.

this window of opportunity analysis using UPPAAL. On the right hand side, the logical formulae verified by the model checker are shown, e.g., `A[] P.data imply x >= 15` that checks that for all possible system states when in the location *data* in the automaton P then the value of the clock x is larger than or equal to 15. This property is verified to be true which is the intended behavior of the system. For more details, we again refer to [16].

6 Conclusion

In this paper we have defined and formalised the *timed decentralised label model*, an extension of the decentralised label model with explicit handling of time through the use of timed automata. We have further shown how this formalism is well-suited for modelling and specifying the security policies for a smart meter system. Finally, we taken the first steps towards using the UPPAAL model checker to automatically verify and validate that the security policies in a system do indeed ensure the relevant security properties.

As future work we plan on investigating if the use of the modeling language Timed Input/Output Automata [9] and the Ecdar [10] tool to more concisely model the semantics of TDLM. As this tool uses a compositional verification method it would potentially allow for checking the policies of much larger TDLM models.

References

1. Alur, R., Dill, D.L.: A theory of timed automata. Theor. Comput. Sci. 126(2), 183–235 (Apr 1994)
2. Anderson, R., Fuloria, S.: On the security economics of electricity metering. In: Proceedings of the 9th Annual Workshop on the Economics of Information Security (WEIS 2010). Cambridge, MA, USA (Jun 2010)
3. Anderson, R., Fuloria, S.: Smart meter security: a survey. Available online at `http://www.cl.cam.ac.uk/~rja14/Papers/JSAC-draft.pdf` (2011)
4. Behrmann, G., David, A., Larsen, K.G.: A tutorial on uppaal. In: Proceedings of the International School on Formal Methods for the Design of Computer, Communication and Software Systems, (SFM-RT 2994). Lecture Notes in Computer Science, vol. 3185, pp. 200–236. Springer (2004)
5. Bell, D.E., LaPadula, L.J.: Secure computer systems: Mathematical foundations. Tech. Rep. ESD-TR-73-278, ESD/AFSC, Hanscom AFB, Bedford, Mass. (Nov 1973)
6. Broberg, N., Sands, D.: Flow locks: Towards a core calculus for dynamic flow policies. In: Proceedings of the 15th European Conference on Programming Languages and Systems (ESOP 2006). Lecture Notes in Computer Science, vol. 3924, pp. 180–196. Springer-Verlag (2006)
7. Broberg, N., Sands, D.: Paralocks: role-based information flow control and beyond. In: Proceedings of the 37th ACM Symposium on Principles of Programming Languages (POPL 2010). pp. 431–444. Madrid, Spain (Jan 2010)
8. Clark, D.D., Wilson, D.R.: A comparison of commercial and military computer security policies. In: Proc. of the IEEE Symposium on Security and Privacy (S&P 1987). pp. 184–194. IEEE (1987)
9. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: Timed I/O automata: a complete specification theory for real-time systems. In: Johansson, K.H., Yi, W. (eds.) Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control, HSCC 2010, Stockholm, Sweden, April 12-15, 2010. pp. 91–100. ACM (2010)
10. David, A., Larsen, K.G., Legay, A., Nyman, U., Wasowski, A.: ECDAR: an environment for compositional design and analysis of real time systems. In: Proceedings of the 8th International Symposium on Automated Technology for Verification and Analysis (ATVA 2010). Lecture Notes in Computer Science, vol. 6252, pp. 365–370. Springer (2010)
11. Molina-Markham, A., Shenoy, P., Fu, K., Cecchet, E., Irwin, D.: Private memoirs of a smart meter. BuildSys 2010 (2010)
12. Myers, A.C.: Mostly-Static Decentralized Information Flow Control. Ph.D. thesis, Massachusetts Institute of Technology (Jan 1999)
13. Myers, A.C., Liskov, B.: A decentralized model for information flow control. In: Proc. of the 16th ACM Symposium on Operating Systems Principles (SOSP 1997). pp. 129–142 (Oct 1997)
14. Myers, A.C., Liskov, B.: Complete, safe information flow with decentralized labels. In: Proc. of the IEEE Symposium on Security and Privacy (S&P 1998). pp. 186–197. IEEE (May 1998)
15. Myers, A.C., Liskov, B.: Protecting privacy using the decentralized label model. ACM Transactions on Software Engineering and Methodology (TOSEM) 9(4), 410–442 (oct 2000)

16. Pedersen, M.L., Sørensen, M.H.: The Timed Decentralised Label Model. Master's thesis, Aalborg University (2015)
17. Sabelfeld, A., Sands, D.: Dimensions and principles of declassification. In: Proceedings of the 18th IEEE Workshop on Computer Security Foundations (CSFW 2005). pp. 255–269. IEEE (2005)