# Conceptualization of a Trust Dashboard for Distributed Usage Control Systems

*Paul Georg Wagner*

Vision and Fusion Laboratory
Institute for Anthropomatics
Karlsruhe Institute of Technology (KIT), Germany
paul.wagner@kit.edu

## Abstract

Achieving data protection and privacy in modern data processing systems is a prominent topic of academic research today. The goal of retaining comprehensive informational sovereignty requires new and innovative solutions, both technological and methodological in nature. Distributed usage control is a popular technology that can give data providers the ability to actively govern the usage of their personal information even in remote systems. However, the architecture of distributed usage control systems is rather complex and often highly dynamic. This makes the assessment of the system's soundness and trustworthiness difficult, especially for untrained laypersons.

In this work we present the concept of a trust dashboard for distributed usage control systems that are backed by trusted computing technologies. The trust dashboard is intended to give users a visual intuition about the current state of the usage control system and its trustworthiness. We achieve this by using a formal model to describe relevant trust dependencies and the actually conducted remote attestations between usage control components, as well as a-priori trust levels for system operators. Based on this we propose a visualization concept that illustrates the current system state and estimates the overall trustworthiness

of the system. Ultimately the trust dashboard aids system operators in the assessment of dynamic and distributed usage control architectures.

# 1 Introduction

Data privacy is one of the major IT-security challenges of our time. Since we live in a world of ubiquitous data acquisition, keeping track of our personal information is an important even though arduous task. This is especially true when personal data are being distributed in highly interconnected and decentralized systems. In the past years, the notion of *data sovereignty* has become prominent in academic research. While no universally accepted definition of (data) sovereignty exists today, it seems clear that retaining ownership of shared information plays an essential role [5]. Hence on our way to data sovereignty, we need to give individuals the possibility to not just track, but control their own private information wherever it may be stored and evaluated.

One technology that can aid in this task is *usage control*. Usage control allows the specification of access rights and obligations that are continuously enforced on a data set throughout its life cycle [9]. Introduced by Park and Sandhu [10] almost two decades ago, a few usage control variants have been developed since then. For one, *distributed usage control* [11] allows the enforcement of usage rules even across system and domain boundaries. This is achieved by operating several independent usage control components in every participating domain. These components then work together over the network to disseminate all usage rules and enforce them simultaneously on all domain systems. Since distributed usage control components have to safeguard critical data and enforce usage rules even on potentially hostile systems, securing them against manipulation and attacks is by no means a trivial task. Most often the integrity of usage control components is protected using *trusted computing* hardware such as Trusted Platform Modules (TPMs). Trusted computing technologies can provide hardware-backed security guarantees that prevent even system owners such as malicious data receivers from tampering with critical parts of the usage control system. With the development of more powerful usage control models, the resulting policy languages and system implementations became increasingly

complicated. Especially in distributed scenarios with many participants, a multitude of usage control components are required for a reliable enforcement of all usage rules. All of this makes it very hard for data owners or even system administrators to decide if the current configuration of a usage control system is safe and if all usage control components are in an acceptable state.

In this work we explore the possibility of using a trust dashboard to aid system owners in the evaluation of distributed usage control systems. This dashboard is intended to give the users a visual intuition about the current state of the usage control system and its trustworthiness. The remainder of this paper is structured as follows. In section 2 we give a brief introduction of distributed usage control systems and describe how they can be protected with trusted computing technologies. Afterwards in section 3 we present a suitable model capable of expressing trust in distributed usage control systems. Based on this model, in section 4 we then define goals and requirements for a trust dashboard and propose a concept for visualizing various states of the usage control system in an intuitive way. We conclude this paper in section 5 with an outlook on future work on this research question.

# 2 Related Work

To prepare for the description of our trust dashboard, we first give a brief introduction of distributed usage control systems and show how trusted computing technologies are typically used to protect them against malicious tampering.

## 2.1 Distributed Usage Control

Usage control (UC) was first proposed in 2002 as a generalization of traditional access control methods [10]. A formal usage control model has been published in 2004 by Park and Sandhu [9]. In general, usage control allows for continuous authorization of data accesses even if the data are already in use. This is in contrast to classical access control schemes, where authorization decisions are made only at the time of the initial data access. Furthermore, usage control supports the declaration of obligations that need to be fulfilled before, during or

after a certain data usage. This is not covered by classical access control either. Ultimately usage control methods can help to describe and enforce complex data usage strategies, such as limiting the number of views or the time of access to sensitive information.

Based on the original usage control model by Park and Sandhu [9], several model variants and formalizations have been developed since [9, 11, 18, 7, 8, 6] One of the most influential improvements is distributed usage control (DUC). Developed by Pretschner et al. [11] in 2006, distributed usage control deals with the enforcement of usage rules even across system and domain boundaries. It allows data providers to specify usage policies in a domain-specific language [4], which are then distributed alongside the sensitive information to any remote data receivers. One way of implementing distributed usage control systems is to rely on a derivative of the XACML reference architecture [13]. Originally developed for attribute-based access control, the XACML components can be canonically extended to implement usage control policies. Figure 2.1 shows a distributed usage control system based on XACML components.

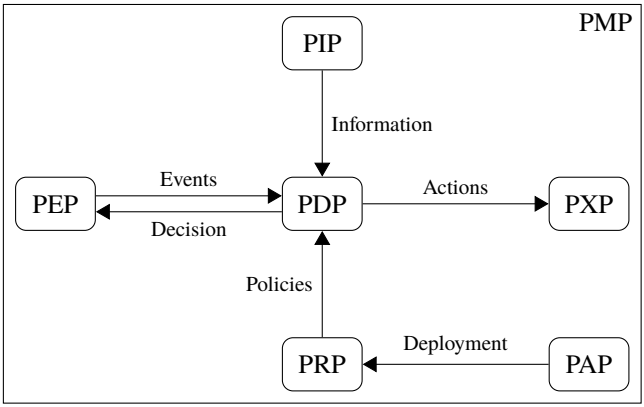



**Figure 2.1**: Distributed usage control components.

At the heart of any distributed usage control system is the *policy enforcement point (PEP)*. PEPs are usually implemented close to data processing applications and are capable of continuously examining and influencing data accesses. Based

on this examination, PEPs broadcast notifications of data usage requests into the usage control system. These notifications are then received and processed by a suitable *policy decision point (PDP)*. PDPs create usage control decisions by evaluating the received notifications against the set of active usage control policies. In addition to the classical binary access decision of `allow` versus `deny`, the PDP can also rule that the data usage described by the event should be `modified` prior to its execution. In the end the PEP receives the decision from its PDP and enforces it on the data processing application.

To aid the PDP in creating correct usage control decisions, two more usage control components are required. The *policy information point (PIP)* can be queried by the PDP for subject and object attributes, as well as generic information such as database entries or environmental properties. The *policy execution point (PXP)* is responsible for executing obligations demanded prior to a data usage, for example the incrementation of an access counter. Obligations are invoked by the PDP and have to be executed successfully before the PDP publishes a positive decision. By utilizing PIP and PXP capabilities, complex and expressive usage control policies can be specified and enforced.

Furthermore there are three auxiliary components involved in the usage control enforcement process. The *policy administration point (PAP)* provides data owners and system administrators an interface to specify and manage usage control policies for their respective data sets. The *policy retrieval point (PRP)* is used by PDPs to retrieve policies from remote usage control systems, e.g. if data accesses to external resources are requested. Finally, the *policy management point (PMP)* controls the distributed usage control system, aids in the lookup of usage control components and facilitates establishing connections between them. In the end it is the collaboration of all components that ensures proper usage control enforcement. Even though the original XACML architecture was intended to specify logical instead of physical system components, in the case of distributed usage control these components are running as dedicated services on different computer systems.

## 2.2 Trusted Computing

Distributed usage control systems allow data providers to restrict access to critical information even after it has been released. However, this only works under the assumption that the participating usage control components are all working correctly. Especially if multiple usage control systems operated by different stakeholders are involved, this assumption is not necessarily sound. For example, remote data receivers may be motivated to tamper with their own usage control components to bypass the enforcement of transmitted policies. Because of this, technical measures have to be taken to prevent malicious system operators from tampering with distributed usage control components.

The proposed solution to this problem is making use of *trusted computing* technologies [3, 2]. Trusted computing allows to protect running software from external influences and verify their integrity by means of hardware-based access control. Among the most widespread trusted computing technologies today are Trusted Platform Modules (TPMs). Usually TPMs consist of a dedicated hardware chip that has been manufactured according to a specification developed by the Trusted Computing Group (TCG) [14]. Many modern desktop and server motherboards already include a TPM hardware chip soldered onto the PCB. Even if no physical TPM is available, software TPMs can be included in the device firmware [12]. However, these firmware modules obviously do not provide the same level of security as their hardware-based counterparts. In general, TPMs are designed to create and hold several cryptographic keys in hardware, which can then be used to encrypt and sign critical information in a secure environment. The private parts of the cryptographic keys stored in the TPM hardware are protected against external influence and cannot be extracted in plain text. Furthermore, TPMs offer *remote attestation* functionality. Remote attestation allows external verifiers to uniquely identify a TPM-equipped computer system and attest to the current state of its software stack. To achieve this, the TPM is used to store unforgeable fingerprints of the current hardware and software configuration in a special set of registers. Then a remote verifier can probe the measured system for proof of its current system state. This is usually done via a cryptographic protocol [17], which establishes a secure channel to the system under test and transmits a so-called *quote*. The quote is a

data structure containing the current system fingerprints and is cryptographically signed by the TPM. The verifier then validates the correctness of the signature and cross-checks the attested fingerprints inside the quote with a set of expected values. If everything validates correctly, the verifier is convinced that the attested fingerprints are correct and the remote system indeed runs a correctly configured and unmodified software stack.

TPM-based remote attestation allows data providers to verify the integrity of remote usage control components before trusting them with enforcing usage rules on their critical data. However, some drawbacks of using TPMs for this purpose have been discovered as well, which can be mitigated by using more powerful trusted computing technologies such as Intel SGX [16]. In any case, conducting remote attestations remains the fundamental instrument of establishing trust in distributed usage control systems.

# 3 Modeling Trust in Usage Control Systems

In order to estimate the trustworthiness of distributed usage control systems, we first require a suitable model. This model should describe existing trust dependencies between usage control components as well as the individually conducted remote attestations at any point in time. For this we partially rely on a model that has been published as part of our previous work [15]. However, for the purpose of defining a trust dashboard we have to extend this base model with the possibility to distinguish different levels of trust and express the trustworthiness of system operators. In the remainder of this section we briefly present the relevant parts from the original publication [15] and then extend the base model to meet our requirements.

## 3.1 Base Model

As described in section 2.1, the basis of a distributed usage control architecture is formed by a set of usage control modules $M$ (e.g. PEP, PDP, ...). Furthermore, we define a set of usage control functions $F$ (e.g. deploy, evaluate, ...). The basic semantic of distributed usage control systems is then specified via a *trust*

*dependency graph* $T = (M, E_T, l_T)$. The edges $E_T \subseteq M \times M$ of this graph describe the trust relationships between the different types of usage control components. More concretely, an edge $(u, v) \in E_T$ means that usage control component $u$ requires an honest component $v$ to perform its task correctly. Finally, the mapping $l_T : E_T \to F$ assigns a human-readable label to each trust dependency, depending on which usage control function is responsible for the dependency. Figure 3.1 shows the trust dependency graph for the XACML-based distributed usage control system presented in section 2.1.

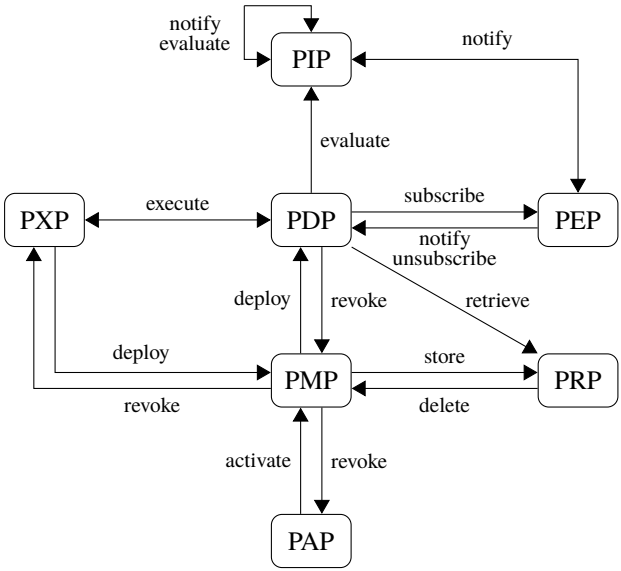



**Figure 3.1**: Trust dependency graph [15].

From the trust dependency graph, an *instance graph* $I = (V_I, E_I, l_I, type_I)$ can be derived. This graph contains concrete instances $V_I$ of usage control components that are being executed, as well as the appropriate trust dependencies $E_I$ and label mappings $l_I$ from the trust dependency graph. Finally, the mapping $type_I : V_I \to M$ assigns a module type to each concrete component instance.

While the trust dependency graph describes the logical architecture of the usage control system, the instance graph describes a concrete realization of it.

Furthermore, we partition the vertices of the instance graph into disjunctive sets called *attestation containers*, denoted by $C \subseteq V_I$. An attestation container describes a set of usage control modules that can be jointly attested. Which module instances form an attestation container depends on the used attestation technology and the system architecture (c.f. section 2.2). The set of all attestation containers is denoted by $\mathcal{C} \subseteq \mathcal{P}(V_I) \setminus \emptyset$. For convenience purposes, we define a function $c_I : V_I \to \mathcal{C}$ that maps a given component to its attestation container. Figure 3.2 shows an example of an instance graph with attestation containers.

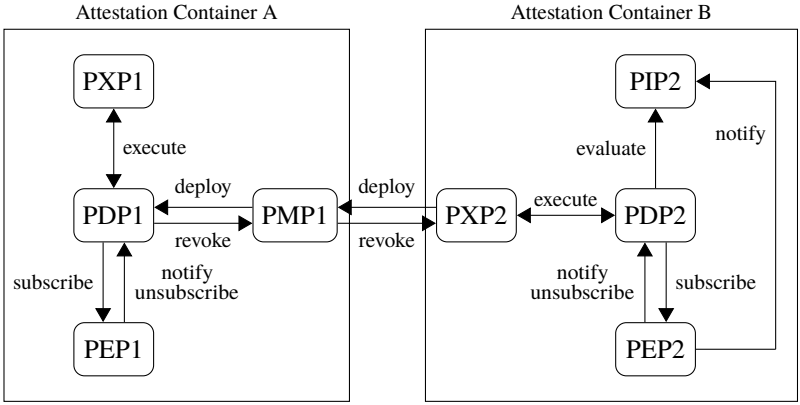



**Figure 3.2**: Instance graph with attestation containers.

Finally, for each component instance $v \in V_I$ we can define the *attestation schedule* $att_v : \mathbb{N}^+ \times \mathcal{C} \to \{-1, 0, 1\}$. The attestation schedule describes the remote attestations that the component $v$ conducts at a certain point in time, and if they are successful (1), not attempted (0) or not successful ($-1$). For more details as well as some examples of the base model we refer the reader to the original publication [15].

## 3.2 Extended Model

To use this model as basis for a trust dashboard, we first extend it with the concept of *operators*. Operators are running usage control components in their own infrastructure and are participating in the distributed usage control system. They can be seen as the administrators responsible for managing the usage control components of their company, or even conceptually as the entire organization itself. Operators can act both as data provider and data receiver, respectively releasing or collecting sensitive information as well as associated usage control policies. We denote the set of operators with $\mathcal{O}$. For each attestation container $C \in \mathcal{C}$ we then identify a single operator $O \in \mathcal{O}$ that is responsible for all usage control components running in the attestation container $C$. We describe this with the operator mapping $o : \mathcal{C} \to \mathcal{O}$. While each attestation container is managed by exactly one operator, a single operator can be responsible for multiple attestation containers at once.

Furthermore, we want to associate a certain level of trust with each operator, and subsequently with the usage control components they are running. For this we define four different trust levels `full`, `marginal`, `untrustworthy` and `unknown`. This categorization is inspired by the trust level definition of PGP [1]. We denote the set of trust levels with $\mathcal{T}$. Finally, we can assign each operator a trust level with the mapping $t : \mathcal{O} \to \mathcal{T}$. This mapping will be customizable by the user of the trust dashboard and serve as basis for including subjective trust assessments into the estimations provided by the dashboard.

# 4 Conceptualizing a Trust Dashboard

In this section we propose a concept for a trust dashboard that helps users to gain insight into the current state of a distributed usage control system by means of an intuitive visual representation. This dashboard is mainly intended to be used by direct participants of the distributed usage control system, such as system operators and/or data providers. The trust dashboard is supposed to assist them in deciding if the distributed system components can be deemed trustworthy, before issuing critical usage control policies or releasing sensitive data.

## 4.1 Goals and Requirements

To present our proposed trust dashboard, we first define the two main design goals of the dashboard and derive appropriate requirements for each of them.

**Goal 1: Visualizing the system state.** The first main goal of our trust dashboard is to give the user an intuitive overview of the current usage control system state. For this, we need to show the user what usage control components are currently running and what dependencies exist between them. Furthermore, we need to illustrate what remote attestations have already been conducted and point out what parts of the trust dependencies have been covered as a result. Ultimately we identify three requirements to achieve this goal.

(Req. 1) Show the user the set of active distributed usage control components, both local and remote (i.e. operated by other participants).

(Req. 2) Show the user the required trust dependencies between the active usage control components.

(Req. 3) Highlight to the user any discrepancies between the required trust dependencies and the actually conducted remote attestations.

**Goal 2: Visualizing the system trustworthiness.** The first goal only aims at expressing the current state of the distributed usage control system in an intuitive way. However, for the trust dashboard this objective alone is not sufficient. We also have to give the user feedback about the resulting trustworthiness of the usage control system in its current state. More concretely, we need to allow the user to specify their subjective estimation of the trustworthiness of certain system operators. Obviously this is highly dependent on the application context, e.g. if a remote system operator is a competitor of the trust dashboard user. Based on this we can then offer some estimates about the trustworthiness of the system overall. In the end goal 2 yields two more requirements to be considered.

(Req. 4) Allow the user to specify their a-priori level of trust in the operators of remote usage control systems.

(Req. 5) Show the user a qualitative estimation of the overall trustworthiness of the usage control system, based on the current attestation schedule and the a-priori trust levels specified by the user.

The remainder of this section shows how we achieve the identified goals and requirements in our proposed dashboard. Finally we illustrate the resulting visual representations of the different trust dashboard states in a small example.

## 4.2 Goal 1: Visualizing the System State

In order to conceptualize a trust dashboard for distributed usage control, we translate the model from section 3 into a simple visualization of the current system state. Starting with an instance graph $I = (V_I, E_I, l_I, type_I)$, as a first step the currently active usage control components $V_I$ are displayed as boxes, labeled with the respective module names derived from the mapping $type_I$. Similarly, the set of operators $\mathcal{O}$ is represented by a number of larger, unfilled boxes around the usage control components. Which components are displayed in which operator boxes is determined by the operator mapping $o : \mathcal{C} \to \mathcal{O}$, depending on the attestation container that the component in question resides in. Together, this satisfies requirement 1. Furthermore, the trust dependency edges $E_I$ between usage control components are visualized with directed arrows connecting the component boxes in the direction of the trust dependency (requirement 2). The states of the trust dependency edges are derived from the current attestation schedules $att_v : \mathbb{N}^+ \times \mathcal{C} \to \{-1, 0, 1\}$ for each usage control component $v \in V_I$ and visualized using different colors for the arrows (requirement 3). Depending on the current time step $t$ and the attestation schedules, we distinguish four different states that a trust dependency edge can be in.

**Recently verified dependency.** A trust dependency edge $(u, v) \in E_I$ is recently verified at time $t$, if there is a successful attestation between $u$ and $v$ not older than $\bar{t}$, and no failed attestation has occurred since. The time interval $\bar{t}$ is a previously defined constant that represents how long a component should be fully trusted after a successful attestation. We mark recently verified trust

dependencies with a green arrow in the trust dashboard. In the formal model, this concept is expressed as equation 4.1.

$$\text{color}(t,u,v) = \text{green} \iff \begin{array}{l} \exists t_s > (t-\bar{t}) : att_u\left(t_s, c_I\left(v\right)\right) = 1 \\ \wedge\, \nexists t_f > t_s : att_u\left(t_f, c_I\left(v\right)\right) = -1 \end{array} \tag{4.1}$$

**Formerly verified dependency.** A trust dependency edge $(u,v) \in E_I$ is formerly verified at time $t$, if there is a successful attestation between $u$ and $v$ older than $\bar{t}$, and neither a failed nor a successful attestation has occurred since. We mark formerly verified trust dependencies with a yellow arrow in the trust dashboard. In the formal model, this concept is expressed as equation 4.2.

$$\text{color}(t,u,v) = \text{yellow} \iff \begin{array}{l} \exists t_s \leq (t-\bar{t}) : att_u\left(t_s, c_I\left(v\right)\right) = 1 \\ \wedge\, \forall \tilde{t} > t_s : att_u\left(\tilde{t}, c_I\left(v\right)\right) = 0 \end{array} \tag{4.2}$$

**Unverified dependency.** A trust dependency edge $(u,v) \in E_I$ is unverified at time $t$, if there are no attestations between $u$ and $v$ so far. We mark unverified trust dependencies with a black arrow in the trust dashboard. In the formal model, this concept is expressed as equation 4.3.

$$\text{color}(t,u,v) = \text{black} \iff \forall \tilde{t} \leq t : att_u\left(\tilde{t}, c_I\left(v\right)\right) = 0 \tag{4.3}$$

**Invalidated dependency.** A trust dependency edge $(u,v) \in E_I$ is invalidated at time $t$, if there is a failed attestation between $u$ and $v$ and no successful attestation has occurred since. We mark invalidated trust dependencies with a red arrow in the trust dashboard. In the formal model, this concept is expressed as equation 4.4.

$$\text{color}(t,u,v) = \text{red} \iff \begin{array}{l} \exists t_f \leq t : att_u\left(t_f, c_I\left(v\right)\right) = -1 \\ \wedge\, \nexists t_s > t_f : att_u\left(t_s, c_I\left(v\right)\right) = 1 \end{array} \tag{4.4}$$

## 4.3 Goal 2: Visualizing the System Trustworthiness

Based on the visualization of the current system state, an estimation for its overall trustworthiness should be given. This is done in two steps. First, the trust

dashboard users define their subjective level of trust in the various operators of usage control components (requirement 4). Similar to before, this is visualized by coloring the operator boxes depending on the assigned level of trust. As described in section 3.2, we distinguish four different trust levels $\mathcal{T}$. We associate the trust level `full` with the color green, trust level `marginal` with the color yellow, trust level `untrustworthy` with the color red and trust level `unknown` with the color black. In terms of the formal model, this step is defining the operator trust mapping $t : \mathcal{O} \to \mathcal{T}$.

Afterwards the trust levels in individual usage control components have to be derived. This is done based on the current state of the trust dependency edges as described in the previous section. For the sake of consistency we use the same trust level categorization for the usage control components as for the operators. Furthermore, the component boxes in the trust dashboard are colored in the same manner as the trust dependency edges and the operator boxes.

**Fully trusted component.** A usage control component $v \in V_I$ is fully trusted at time $t$ if there is at least one recently verified and no invalid trust dependency to $v$. We mark fully trusted components with a green box in the trust dashboard. In the formal model, this concept is expressed as equation 4.5.

$$\text{color}(t, v) = \text{green} \iff \begin{array}{l} \exists u \in V_I : \text{color}(t, u, v) = \text{green} \\ \wedge \nexists u \in V_I : \text{color}(t, u, v) = \text{red} \end{array} \tag{4.5}$$

**Marginally trusted component.** A usage control component $v \in V_I$ is marginally trusted at time $t$ if there is at least one formerly verified trust dependency to $v$, but no recently verified or invalid ones. We mark marginally trusted components with a yellow box in the trust dashboard. In the formal model, this concept is expressed as equation 4.6.

$$\text{color}(t, v) = \text{yellow} \iff \begin{array}{l} \exists u \in V_I : \text{color}(t, u, v) = \text{yellow} \\ \wedge \forall u \in V_I : (\text{color}(t, u, v) = \text{yellow} \\ \vee \text{color}(t, u, v) = \text{black}) \end{array} \tag{4.6}$$

**Untrusted component.** A usage control component $v \in V_I$ is untrusted at time $t$ if there is at least one invalid trust dependency to $v$. We mark untrusted components with a red box in the trust dashboard. In the formal model, this concept is expressed as equation 4.7.

$$\text{color}(t, v) = \text{red} \iff \exists u \in V_I : \text{color}(t, u, v) = \text{red} \tag{4.7}$$

**Component with unknown trust level.** A usage control component $v \in V_I$ has an unknown trust level at time $t$ if there are only unverified trust dependencies to $v$. We mark components of unknown trust level with a black box in the trust dashboard. In the formal model, this concept is expressed as equation 4.8.

$$\text{color}(t, v) = \text{black} \iff \forall u \in V_I : \text{color}(t, u, v) = \text{black} \tag{4.8}$$

Once the operator trust mapping and the trust level of individual components is defined, an overall trust estimation should be derived from it (requirement 5). For simplicity we use a qualitative estimation with three trust levels that are each visualized with a distinct icon in the trust dashboard.

**Trusted overall system state.** A distributed usage control system with instance graph $I$ is in a trusted state at time $t$ if all usage control components of $I$ are either fully trusted, or belong to a system operator that is fully trusted by the trust dashboard user. Broadly speaking, this means that the usage control system has verified all doubtful components with a recently conducted remote attestation. In the trust dashboard, this overall system state is visualized by a green checkmark next to the trust graph. In the formal model, it is expressed as equation 4.9.

$$\text{state}(t) = \text{trusted} \iff \begin{array}{c} \forall v \in V_I : \text{color}(t, v) = \text{green} \\ \vee\, t\left(o\left(c_I\left(v\right)\right)\right) = \text{full} \end{array} \tag{4.9}$$

**Ambiguous overall system state.** A distributed usage control system with instance graph $I$ is in an ambiguous state at time $t$ if it is not trusted, but no untrusted components exist in $I$ either. This means that not every potentially dangerous usage control component has been cryptographically verified, but

there is no evidence for malicious behavior. In the trust dashboard, this overall system state is visualized by a yellow questionmark next to the trust graph. In the formal model, it is expressed as equation 4.10.

$$\text{state}(t) = \text{ambiguous} \iff \begin{array}{l} \text{state}(t) \neq \text{trusted} \\ \wedge \forall v \in V_I : \text{color}(t, v) \neq \text{red} \end{array} \tag{4.10}$$

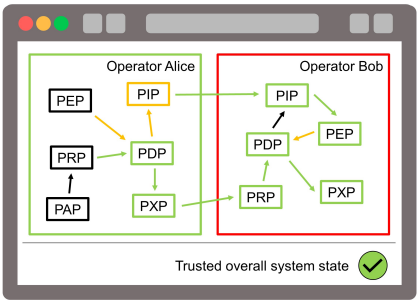
**Untrusted overall system state.** A distributed usage control system with instance graph $I$ is in an untrusted state at time $t$ if there is at least one untrusted component in $I$. This means that at least one usage control component has failed the verification via a remote attestation, and hence we have to assume malicious interference in the usage control system. In the trust dashboard, this overall system state is visualized by a red crossmark next to the trust graph. In the formal model, it is expressed as equation 4.11.

$$\text{state}(t) = \text{untrusted} \iff \exists v \in V_I : \text{color}(t, v) = \text{red} \tag{4.11}$$

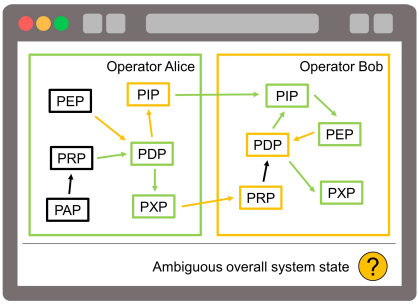
## 4.4 Trust Dashboard Examples

To illustrate the nature of the resulting visualization, in figure 4.1 we give a few simple examples of the trust dashboard for all three system states. For this we assume there to be two operators Alice and Bob, each managing several usage control components. In our examples we show the trust dashboard from the point of view of operator Alice, so we assume her to always be a fully trusted operator. As described in section 4.3, this is indicated by a green operator box around her components.

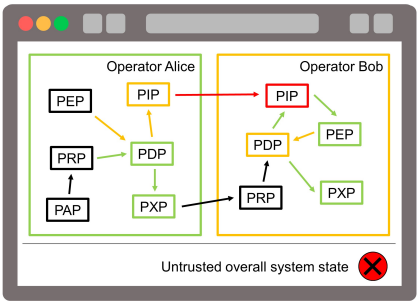
The first example in figure 4.1(a) shows the dashboard layout in a trusted overall system state. This is despite operator Bob being untrusted in this example, as indicated by the red operator box around his components. Since all of Bob's components have at least one recently verified incoming dependency (green arrows), they are all classified as fully trusted and marked with green boxes (see eq. 4.5). As a result, all components of the system are either fully trusted or belong to a fully trusted operator. With eq. 4.9 follows that the overall system state is trusted. In the second example in figure 4.1(b), Bob's PRP only has a




(a) Trusted system state.




(b) Ambiguous system state.




(c) Untrusted system state.

**Figure 4.1**: Examples of the trust dashboard visualization in the three system states.

formally verified dependency (yellow arrow), most likely because the previously conducted attestation timed out. Because of this, Bob's PRP is left as only marginally trusted (see eq. 4.6) and the overall system state results to ambiguous, as described in eq. 4.10. The last example in figure 4.1(c) shows an invalidated dependency between Alice's PIP and Bob's PIP (red arrow). This invalidation is due to a failed remote attestation between both PIPs, which results in Bob's PIP being classified as untrusted (see eq. 4.7) and marked with a red box. Because of this deterioration in trust, according to eq. 4.11 now the entire system has to be seen as untrustworthy as well.

# 5 Conclusion

In this work we presented the concept of a trust dashboard for distributed usage control systems that are backed by trusted computing technologies. Based on a formal model describing generic usage control systems and the relevant trust dependencies, we proposed a visualization concept that (i) illustrates the current system state and (ii) estimates the overall trustworthiness of the system. To achieve the first goal, we classified the trust dependencies between distributed usage control components based on the recency and outcome of conducted remote attestations. By combining the current system state with a-priori trust levels for system operators, we then provided the dashboard user with a qualitative estimation of the overall system trustworthiness. Ultimately our approach contributes a tool to help usage control system operators in the assessment of dynamic and distributed system architectures.

In the future we plan to implement our concept as a web-based application and deploy it to real-world usage control systems. Based on this, user studies can be conducted to evaluate the benefits of the trust dashboard in real-world scenarios. Furthermore we are aware of some issues that remain unaddressed with our approach. So far the specific trusted computing technologies protecting the usage control components are not being considered in either the formal model or the trust dashboard concept. However, since different technologies yield different benefits and drawbacks, clearly there are opportunities to refine and improve both the visualization of the current system state, as well as the

resulting trust estimation. In addition, the current trust estimation method is based on a simple qualitative heuristic. To obtain better trust estimation results, a more sophisticated approach based on probabilistic estimations can be undertaken. This would result in a quantitative (albeit still subjective) trust estimation methodology.

# References

[1] Alfarez Abdul-Rahman. "The pgp trust model". In: *EDI-Forum: the Journal of Electronic Commerce*. Vol. 10. 3. 1997, pp. 27–31.

[2] Masoom Alam et al. "Model-based behavioral attestation". In: *Proceedings of the 13th ACM symposium on Access control models and technologies*. 2008, pp. 175–184.

[3] Agreiter Berthold et al. "A technical architecture for enforcing usage control requirements in service-oriented architectures". In: *Proceedings of the 2007 ACM workshop on Secure web services*. 2007, pp. 18–25.

[4] Manuel Hilty et al. "A policy language for distributed usage control". In: *European Symposium on Research in Computer Security*. Springer. 2007, pp. 531–546.

[5] Patrik Hummel et al. "Data sovereignty: A review". In: *Big Data & Society* 8.1 (2021), p. 2053951720982012.

[6] Radha Jagadeesan et al. "Timed constraint programming: a declarative approach to usage control". In: *Proceedings of the 7th ACM SIGPLAN international conference on Principles and practice of declarative programming*. 2005, pp. 164–175.

[7] Helge Janicke, Antonio Cau, and Hussein Zedan. "A note on the formalisation of UCON". In: *Proceedings of the 12th ACM symposium on Access control models and technologies*. 2007, pp. 163–168.

[8] Fabio Martinelli and Paolo Mori. "A Model for Usage Control in GRID systems". In: *2007 Third International Conference on Security and Privacy in Communications Networks and the Workshops-SecureComm 2007*. IEEE. 2007, pp. 169–175.

[9] Jaehong Park and Ravi Sandhu. “The UCON ABC usage control model”. In: *ACM Transactions on Information and System Security (TISSEC)* 7.1 (2004), pp. 128–174.

[10] Jaehong Park and Ravi Sandhu. “Towards usage control models: beyond traditional access control”. In: *Proceedings of the seventh ACM symposium on Access control models and technologies*. 2002, pp. 57–64.

[11] Alexander Pretschner, Manuel Hilty, and David Basin. “Distributed usage control”. In: *Communications of the ACM* 49.9 (2006), pp. 39–44.

[12] Himanshu Raj et al. “ftpm: A software-only implementation of a {TPM} chip”. In: *25th {USENIX} Security Symposium ({USENIX} Security 16)*. 2016, pp. 841–856.

[13] OASIS Standard. *extensible access control markup language (xacml) version 3.0*. 2013.

[14] *TCG Specification Architecture Overview. Specification Revision 1.4*. `https://trustedcomputinggroup.org/wp-content/uploads/TCG_1_4_Architecture_Overview.pdf`. Accessed: 2021-11-16.

[15] Paul Georg Wagner. “Towards a Formal Model for Quantifying Trust in Distributed Usage Control Systems”. In: *Proceedings of the 2019 Joint Workshop of Fraunhofer IOSB and Institute for Anthropomatics, Vision and Fusion Laboratory*. 2019, p. 113.

[16] Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. “Distributed usage control enforcement through trusted platform modules and sgx enclaves”. In: *Proceedings of the 23nd ACM on Symposium on Access Control Models and Technologies*. 2018, pp. 85–91.

[17] Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. “Establishing Secure Communication Channels Using Remote Attestation with TPM 2.0”. In: *International Workshop on Security and Trust Management*. Springer. 2020, pp. 73–89.

[18] Xinwen Zhang. *Formal model and analysis of usage control*. George Mason University, 2006.