Towards a Formal Model for Quantifying Trust in Distributed Usage Control Systems

Paul Georg Wagner

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

Technical Report IES-2019-06

Abstract

Distributed usage control is a form of usage control that spans over multiple domains and computer systems. As a result, usage control components responsible for evaluating policies, gathering information, executing actions and enforcing decisions are operated in the vicinity of different stakeholders with conflicting interests. In order to prevent malicious stakeholders from manipulating these components, remote attestation can be used to verify the integrity of their code base. However, in a distributed case it is not always apparent what sequence of attestations is necessary and which verifier should conduct them. Furthermore, it is unclear what impact a failed attestation has on the trustworthiness of the whole usage control system. To solve these questions, it is necessary to identify which agents need to trust each other in order to securely execute a certain usage control function. Then the sequence of remote attestations that occur across the distributed usage control system can be examined accordingly. In this work we develop a formal model that represents the trust relationships of distributed usage control systems with multiple collaborating actors. Based on the conducted attestations we define simple binary and non-binary trust metrics that quantify the trust level a data owner can expect at a certain point in time.

Finally we show how the model can be used to determine the level of trust reached in a real-world scenario.

1 Introduction

In recent years, usage control (UC) has been more and more propagated as a novel technology for governing access to valuable information. Unlike classical access control, usage control models focus on managing the future usage of data [7]. With usage control technology it is possible to restrict access to protected assets even after they have been disclosed. Often usage control is used in distributed environments, where sensitive data are shared between shareholders. One such example is the Fraunhofer research project *International Data Space* [6]. The International Data Space allows data providers to distribute valuable data alongside usage restrictions to potentially malicious data consumers. The data consumer's systems then process the received information according to the published rules. Naturally, the data provider wants to ensure that the data consumer can be trusted to obey the issued usage restrictions on his data. For this the International Data Space uses distributed UC modules that independently evaluate the usage control policies and enforce the resulting decisions. Since each participant of the data space may act maliciously and try to extract foreign data past the protection mechanisms, it is necessary to verify the integrity of the UC components prior to the data exchange.

Trusted computing is the state of the art approach that allows for remote verification of software components. Currently the most widespread trusted computing technologies are Trusted Platform Modules (TPMs) [9] and Intel's Software Guard Extensions (SGX) [3]. Both of these technologies support establishing trust in remote software stacks by verifying code bases through special hardware and cryptographic methods. This software verification process is called *remote attestation*. Besides verifying the integrity of a software stack, remote attestation also establishes secure channels between prover and verifier. The International Data Space uses TPMs and a customized remote attestation protocol to establish trust in data consumers. However, when developing distributed usage control systems that establish trust by remote attestation, several open questions remain. For example, it is not always clear which components have to be attested, and by which verifiers. Comprehensive usage control systems are complicated and security relevant UC functions may span over multiple distributed UC components. This is especially true if the usage control system also includes components that track and store the provenance of supervised data. In these cases it has to be ensured that all involved UC components are properly attested and can securely communicate with each other. Another interesting question is what impact a failed attestation has on the security of the overall system. These questions all emerge from the yet unsolved problem of quantifying the trust propagation in dynamically operating and distributed usage control systems.

In this work we develop a formal model that can represent the trust relationships that occur in distributed usage control systems with multiple collaborating actors. This model is independent from the design of the UC-system, its implementation, and the used trusted computing technology. Furthermore we define simple binary and non-binary trust metrics that can be used to determine the trust level of certain UC functionalities at a specific point in time. Calculating a dynamic trust level for a UC system is very beneficial for conducting a comprehensive security analysis of the infrastructure. Finally we show how the model can be used to determine the level of trust in a real-world example scenario based on the International Data Space using TPM-based attestation.

2 Related work

Managing and distributing trust has been a major topic of research interest for a long time. By far the most widespread technique of managing trust in distributed systems is via a *public key infrastructure* (PKI). With a PKI, a few trusted certification authorities (CAs) issue signed public keys for the agents in their domain. As a result, the trust in a certain communication channel is reduced to the trustworthiness of the CA. Even though PKIs are a fundamentally important concept in IT security, as a centralized way of managing trust they are not applicable to our scenario. In terms of decentralized approaches to trust management, the most important concept is the *Web of Trust* [1], which has been popularized by the well-known PGP software. Its main principle is to distribute trust transitively by endorsement of already trustworthy collaborators (i.e. “my friend’s friend is my friend”). Also, it is possible to generate new trust by offline comparison of public key fingerprints. This decentralized version of trust distribution already comes close to what we need for our scenario. A usage control component could determine the level of trust in a remote system based on the trust that their peers already have in it. New trust would then be generated by automated remote attestation instead of manually comparing fingerprints. However, the Web of Trust does not offer any kind of trust metric, and does not take possible internal attackers into account. Also it does not give any notion of time.

An approach that factors in these aspects are dynamic reputation systems [5, 4, 10]. Their idea is to describe trust mathematically and develop a metric for the reputation of an agent based on their previous behavior. Simply put, if an agent behaves cooperatively, its level of trust increases. If the agent defects, the trust level is impacted. However, since it is not at all well-defined what constitutes as “cooperative behavior” in our scenario, reputation systems also do not suffice for quantifying trust in distributed UC systems. Furthermore, they neither define what actions are suitable to increase or decrease trust, nor do they deal with attestation mechanisms. Since our goal is to develop a formal model of distributed UC systems that works independently of the system design or the used attestation technology, reputation systems do not meet our requirements.

3 Formal model

Our goal is to develop a metric that quantifies the level of trust in distributed UC systems. For this, a formal model is required that describes the trusted communication between usage control components. Since trust relationships can be intuitively modeled as graphs, we utilize a graph-based approach. Furthermore, the formal model needs to represent attestations conducted by the UC components as well as the architecture of the deployed UC system. In this section, we develop a suitable model in three steps.

1. Define functions of the UC components that have to be trusted using a graph-based model (global).
2. Define the existing agents and cross-system activities of interest by instantiating that graph (scenario specific).
3. Define the system architecture by binding the agents to attestable systems (implementation specific).

As a first step, the basic semantic of the usage control system is specified via a *trust dependency graph*. The trust dependency graph contains the existing types of usage control components. It describes how they need to trust each other for any interaction that may occur between them. In the second step we concretize this graph by considering the actual components that are operated in the distributed usage control system. For this we represent each concrete UC component as an instance of a node from the trust dependency graph. We call a concrete UC component *agent*, because it needs to securely interact with other components in the system. The resulting graph is called *agent graph*. Unlike the trust dependency graph, each agent graph is specific to a certain scenario that the UC system is deployed for. It also yields information about the actors that operate the usage control components in that scenario. The agent graph can be partitioned into multiple *UC activities*, which represent a function of the distributed UC system spanning over multiple UC components. We will later show how the trust level of a UC activity can be measured using an instance of the model. Finally, an *architecture graph* defines how the agents map to physical computer systems that can be attested. The architecture graph is not only specific for a certain UC scenario, but also depends on the used trusted computing technology and the deployment of UC components. Figure 3.1 shows an overview of the steps required to transfer the design and implementation of a UC system into the formal model. In the following sections we present this formal model in detail. Afterwards we develop trust metrics that can be evaluated on an instance of the model.

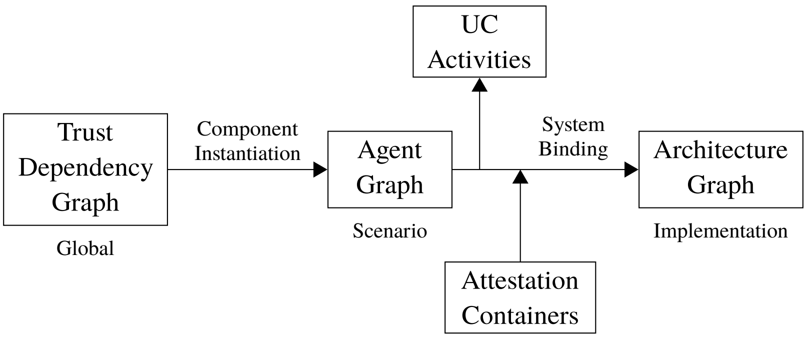

Figure 3.1: Overview of the formal model.

3.1 Defining UC systems

The first step of the formal model includes defining a distributed UC system and its components. This is done in definition 3.1.1.

Definition 3.1.1 (DUC system)**.** Let M be a finite set of DUC modules and F a set of DUC functions. We call the tuple $S := (M, F)$ *DUC system.*

Besides the UC components and their functions, we also need to define how the UC components may interact with each other. This is done by the trust dependency graph, as described in definition 3.1.2.

Definition 3.1.2 (Trust Dependency Graph)**.** Let $S = (M, F)$ be a DUC system. Let $E^F \subseteq M \times M$ be a set of directed edges over M and $l^F : E^F \to F$ a mapping that labels each edge with a system function. We call the triple $FG := \left(M, E^F, l^F\right)$ *trust dependency graph* of S.

The trust dependency graph of a DUC system describes the inter-component functions that may be called across the distributed system. A trust dependency graph can be constructed solely with knowledge of the UC component's interfaces. It is not necessary to know the use case or the usage control policies that should be deployed. Hence the trust dependency graph is independent of the system's concrete realization and implementation.

An example for a trust dependency graph is presented in figure 3.2. It shows the trust dependency graph for the XACML-based distributed usage control architecture that is deployed in the International Data Space. XACML [2] is a reference architecture that defines usage control components responsible for enforcement (PEP), policy evaluation (PDP), information gathering (PIP), and administration (PAP). Besides these XACML-based components, the usage control architecture of the International Data Space uses some additional components responsible for retrieving policies (PRP), managing communication (PMP) and executing obligations (PXP). The displayed DUC system is modeled as $M = \{PEP, PDP, PIP, ...\}$ and $F = \{notify, evaluate, execute, ...\}$. The trust dependency graph shows the possible interactions and the resulting trust dependencies between components as labeled edges. Note that the direction of

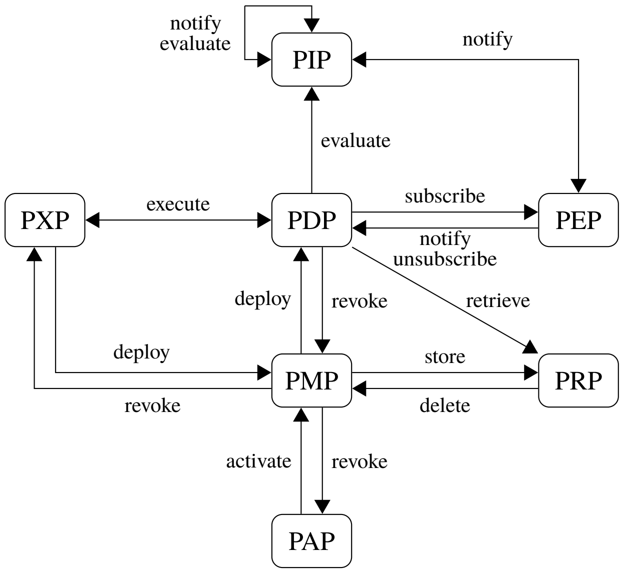

Figure 3.2: Example of a trust dependency graph.

the edge defines the direction of the trust dependency, which does not always correspond with the direction of the interaction. For example, a PAP may revoke a policy by calling the *revoke* function of the responsible PMP. However, the edge is directed the opposite way, because in this case the PMP has to trust the PAP that the revocation request is legit.

3.2 Defining agents and activities

In order to represent a specific scenario, we can instantiate the trust dependency graph and introduce agents that interact with each other. This is done in definition 3.2.1.

Definition 3.2.1 (Agent Graph)**.** Let $FG = \left(M, E^F, l^F\right)$ be a trust dependency graph. Let A be a set of agents, $E \subseteq A \times A$ a set of directed edges over A and $l : A \to F$ a mapping. Let also be $type : A \to M$ a mapping that assigns a module type to each agent. We call the tuple $G := (A, E, l, type)$ *agent graph*, if it holds that

$$\forall(a, b) \in E : (type(a), type(b)) \in E^F$$
$$\forall(a, b) \in E : l\,(a, b) = l^F\,(type(a), type(b))$$

According to definition 3.2.1, every agent is an instance of a UC component. The agent interaction corresponds to the DUC functions that have been described by the trust dependency graph. The two conditions in 3.2.1 ensure that the agent graph only contains edges that correspond to the trust dependency graph (i.e. agents can only call existing functions). Note that the agent graph may contain multiple agents of one particular type (e.g. if multiple PIPs or PEPs exist), while the trust dependency graph contains each component exactly once.

The agent graph shown in figure 3.3 is based on the example trust dependency graph in figure 3.2. The example agent graph shows a scenario with two actors A and B, who operate distributed usage control components. In this scenario, the PXP instance of actor B is responsible for deploying policies at the PDP instance of actor A. This allows B to enforce usage control policies on his data, even if they are shared with A. Note that the agent graph contains multiple instances of a single UC component. For example, in this case both actors A and B operate PDPs, PEPs and PXPs. While the trust dependency graph is of a global nature and represents an abstract DUC architecture, agent graphs

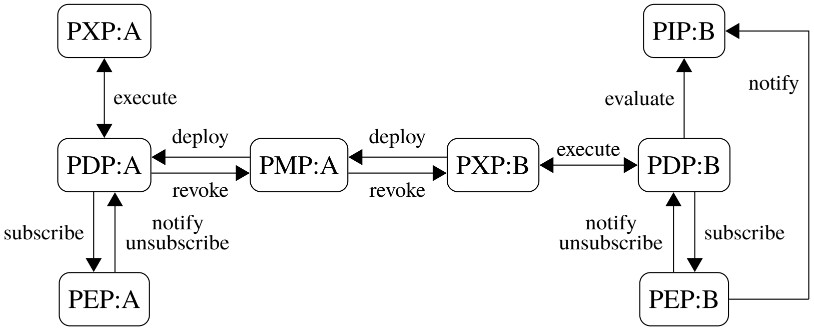

Figure 3.3: Example of an agent graph.

derived from it depend on specific use cases. Also note that all agent graph edges correspond to edges from the trust dependency graph, but not all trust relationships may be included in the agent graph, depending on their relevance for the scenario.

Besides defining the agents of the UC system, we also have to specify what kind of agent interaction should be evaluated for trustworthiness. Definition 3.2.2 partitions the agent graph into multiple acyclic subgraphs called *UC activities*. A UC activity represents an action that requires multiple agents to work together, such as the deployment of policies or the enforcement of access decisions. Since the involved agents have to trust each other in order to reliably execute these actions, the trust level of a UC system will be based on the relevant UC activities.

Definition 3.2.2 (UC Activity)**.** Let $G = (A, E, l, type)$ be an agent graph. Let $H := (\bar{A}, \bar{E}, \bar{l})$ be a connected subgraph of G with $\bar{A} \subseteq A$, $\bar{E} \subseteq E \cap \left(\bar{A} \times \bar{A}\right)$ and $\bar{l} := l|_{\bar{E}}$. We call the subgraph H *UC activity* of G, if

$$\begin{aligned} H &\text{ is acyclic} \\ \exists! x \in \bar{A} &: indeg(x) = 0 \\ \exists y \in \bar{A} &: outdeg(y) = 0 \end{aligned}$$

The unique vertex x is called *root* of H. A vertex y is called *leaf* of H. The set of all leaves is denoted by Y.

Figure 3.4 shows an example for a UC activity based on the agent graph in figure 3.3. The depicted UC activity represents the necessary interaction for locally enforcing a policy. First, the enforcement point (PEP) notifies the decision point (PDP) of an access request. The PDP then evaluates the policies, requests necessary information at the PIP and executes obligations at the PXP. In this activity the PEP acts as root, while the PXP and the PIP are leaves. In order to trust the UC activity of local policy enforcement, all of these interactions need to be secure. Complex distributed usage control systems, such as the International Data Space, have many more relevant UC activities that can be identified, including remote policy enforcement, policy deployment, and policy revocation. However, for the remainder of this paper we will stick to the example of local policy enforcement.

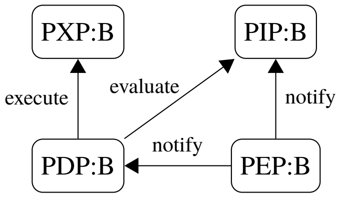

Figure 3.4: Example of a UC activity: Local policy enforcement.

3.3 Defining attestations and architectures

Finally the formal model needs to contain information about the remote attestations that can be executed by the agents. In order to accommodate this, definition 3.3.1 introduces the notion of *attestation containers*. An attestation container is a set of agents that can be jointly attested. Which agents form an attestation container depends on the used attestation technology and the system architecture. For example, if the UC system uses TPMs to execute the remote attestations, all UC components running on a TPM-protected computer system are included in an attestation container. More advanced trusted computing technologies, such as Intel's SGX, allow the attestation of software enclaves rather than whole computer systems. In that case, all UC components included inside such an enclave form an attestation container. The tuple of agent graph and attestation containers is called *architecture graph*.

Definition 3.3.1 (Attestation Container)**.** Let $G = (A, E, l, type)$ be an agent graph. We call a non-empty set $C \subseteq A$ *attestation container*, if all $c \in C$ can be jointly attested. The set of all attestation containers is denoted by $\mathcal{C} \subseteq \mathcal{P}(A) \setminus \emptyset$. The tuple $(G, \mathcal{C})$ is called *architecture graph*.

Based on the description of the attestation container, we have to represent the concrete attestations that agents actually conduct during runtime. This is done in definition 3.3.2 via an *attestation schedule*.

Definition 3.3.2 (Attestation Schedule)**.** Let $(G, \mathcal{C})$ be an architecture graph. For any agent $a \in A$ we call the mapping $att_a : \mathbb{N}^+ \times \mathcal{C} \to \{-1, 0, 1\}$ *attestation schedule*. The family of all attestation schedules is denoted by $\mathcal{A} = (att_a)_{a \in A}$.

The attestation schedule of an agent indicates which attestations the agent conducts at what points in time, and if they are successful. More concretely, if $att_a(t, C) = 1$, then at time t the agent a conducts a successful remote attestation of container C. This means that a successfully verifies the integrity of all agents that are included in C. If instead $att_a(t, C) = -1$, the attestation fails and the agent is unable to verify the integrity of C. If $att_a(t, C) = 0$, the agent a does not conduct a remote attestation of container C at time t.

4 Quantifying trust

The formal model allows us to mathematically represent a distributed UC system. Based on an architecture graph and the associated attestation schedules we can now define trust metrics for the relevant UC activities.

4.1 Binary trust metrics

Given a UC activity H, we denote the level of trust in the activity at time t by $TrustLevel^H(t) \in \{0, 1\}$. A trust level of 1 means that the activity is trusted, while a trust level of 0 indicates that the attestations are not sufficient to ensure the integrity of all involved components. In order to define this trust level, we are examining the paths of H and calculate trust gains for each transition within the path.

4.1.1 Trust gain by attestations

Whenever an agent a conducts a successful attestation of container C, possible transitions between a and another agent $c \in C$ are trusted and positively influence the trust level of H. However, this positive influence only lasts as long as no other agent unsuccessfully conducts an attestation of C, thereby determining that its integrity cannot be trusted anymore. This idea is expressed in definition 4.1.1. Like the overall trust level, the trust gain is binary. A trust gain of 1 for the transition (a, b) means that b has been attested by a, and no other agent failed in verifying the integrity of b since. A trust gain of 0 indicates that a has not yet attested a container that includes b, or that such an attestation is outdated.

Definition 4.1.1 (Trust Gain by Attestation)**.** Let $(G, \mathcal{C})$ be an architecture graph and $\mathcal{A} = (att_a)_{a \in A}$ the family of associated attestation schedules. Let H be a UC activity of G and $(v_1, ..., v_n) \in H$ a path of the activity. The trust gain by attestation for the transition (v_{i-1}, v_i) at time t is defined as

$$Gain^{att}(i,t) := \begin{cases} & \exists C \in \mathcal{C}, t_1 \leq t : \\ 1, & \quad v_i \in C \wedge att_{v_{i-1}}(t_1, C) = 1 \wedge \\ & \quad \forall a \in A : \nexists t_2 \in [t_1, t] : att_a(t_2, C) = -1 \\ 0, & \text{else} \end{cases}$$

4.1.2 Trust gain by locality

While it is clear that attesting a remote component increases trust, we also have to manage the trust gains of local components. If two dependent UC components are included in the same attestation container, they can communicate securely without conducting a remote attestation. However, even though a remote attestation is not required for establishing a secure channel, the integrity of both components still needs to be verified. Hence we have to demand that a previous component attests both of the local components. This concept of trust gain by locality is specified in definition 4.1.2.

Definition 4.1.2 (Trust Gain by Locality)**.** Let $(G, \mathcal{C})$ be an architecture graph and $\mathcal{A} = (att_a)_{a \in A}$ the family of associated attestation schedules. Let H be a UC activity of G and $(v_1, ..., v_n) \in H$ a path of the activity. The trust gain by locality for the transition (v_{i-1}, v_i) at time t is defined as

$$Gain^{loc}(i,t) := \begin{cases} 1, & \begin{aligned} &\exists C \in \mathcal{C}, t_1 \leq t : \\ &\qquad \{v_{i-1}, v_i\} \subseteq C \wedge \\ &\qquad \exists j < i : att_{v_j}(t_1, C) = 1 \wedge \\ &\qquad \forall a \in A : \nexists t_2 \in [t_1, t] : att_a(t_2, C) = -1 \end{aligned} \\ 0, & \text{else} \end{cases}$$

4.1.3 Putting it together

Given the two concepts of generating trust in a distributed UC system, we can define the trust level for a UC activity. We can base the definition on the trust gain by attestation, the trust gain by locality, or both. Definition 4.1.3 specifies the trust level of a path by multiplying the trust gains of the respective transitions. The trust level of the whole UC activity is the minimal trust over all paths.

Definition 4.1.3 (Trust Level)**.** Let $(G, \mathcal{C})$ be an architecture graph and further let $H = (\bar{A}, \bar{E}, \bar{l})$ a UC activity of G with root $x \in \bar{A}$ and leaves $Y \subseteq \bar{A}$. The trust level of a path $(v_1, ..., v_n) \in H$ is defined as

$$TrustLevel^{(v_1, \ldots, v_n)}(t) := \prod_{i=2}^{n} (Gain(i,t))$$

Depending on the scenario, the trust gain is defined by attestation or attestation and locality.

$$\begin{aligned} Gain(i,t) &:= Gain^{att}(i,t) \\ Gain(i,t) &:= \max(Gain^{att}(i,t), Gain^{loc}(i,t)) \end{aligned}$$

The trust level of the UC activity H is defined as

$$TrustLevel^{H}(t) := \min_{\substack{(v_1, \ldots, v_n) \in H \\ v_1 = x, v_n \in Y}} \left(TrustLevel^{(v_1, \ldots, v_n)}(t) \right)$$

Note that the trust level definition is based on the transitions between agents in the UC activity, instead of the agents themselves. Unlike many existing reputation systems (c.f. section 2), we do not define the trust level of a certain agent at all. Instead we define the trust gain of a transition within a UC activity, and then generalize that definition over paths to the whole activity. The reason for this is that remote attestation is not only responsible for verifying the integrity of agents, but also establishes a secure channel for communication. Hence it is not sufficient to focus just on the level of trust in the agent, we need to examine the connections between them.

4.2 Non-binary trust metrics

A binary trust metric can only distinguish trusted from untrusted systems. In order to quantify trust more precisely, we can define non-binary trust metrics. In that case, given a UC activity H, we denote the level of trust in the activity at time t by $\mathit{TrustLevel}^H(t) \in [0, 1]$.

A simple non-binary trust metric can be obtained by including the temporal decay of trust in the model. For this we introduce a dampening factor $\eta : \mathbb{N}_0^+ \to [0, 1]$ and modify the definitions of trust gains.

Definition 4.2.1 (Trust Gains with Temporal Decay)**.**

$$\mathit{Gain}^{att}(i, t) := \begin{cases} & \exists C \in \mathcal{C}, t_1 \leq t : \\ \eta(t - t_1), & \quad v_i \in C \land \mathit{att}_{v_{i-1}}(t_1, C) = 1 \land \\ & \quad \forall a \in A : \nexists t_2 \in [t_1, t] : \mathit{att}_a(t_2, C) = -1 \\ 0, & \text{else} \end{cases}$$

$$\mathit{Gain}^{loc}(i, t) := \begin{cases} & \exists C \in \mathcal{C}, t_1 \leq t : \\ & \quad \{v_{i-1}, v_i\} \subseteq C \land \\ \eta(t - t_1), & \quad \exists j < i : \mathit{att}_{v_j}(t_1, C) = 1 \land \\ & \quad \forall a \in A : \nexists t_2 \in [t_1, t] : \mathit{att}_a(t_2, C) = -1 \\ 0, & \text{else} \end{cases}$$

The definition of the dampening factor η depends on the scenario. In general, the choice of η reflects how fast the generated trust deteriorates after a successful attestation. For most cases a polynomial or exponential decay should be an adequate choice.

$$\eta(t) := (t+1)^{-p}$$
$$\eta(t) := \exp(-\lambda t)$$

4.3 Example calculation

After defining binary and non-binary trust metrics, we give an example calculation based on the previously used International Data Space scenario. For this, we take the UC activity representing local policy enforcement from figure 3.4 and define suitable attestation containers. As shown in figure 4.1, the set of attestation containers results to $\mathcal{C} = \{\{pip\}, \{pdp, pxp\}\}$. Since the International Data Space uses TPMs to provide proof of integrity during remote attestation, in this case the attestation containers represent physical computer systems.

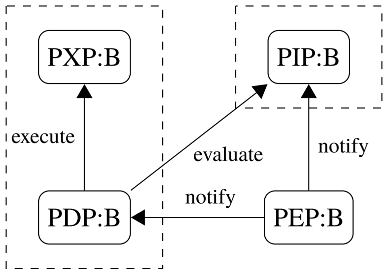

Figure 4.1: UC activity with attestation containers: Local policy enforcement.

In order to determine the trust level of this scenario, we have to specify the attestation schedule. We assume that the PXP and the PIP do not conduct any attestations in this example.

$$\forall t \in \mathbb{N}^+, C \in \mathcal{C} : att_{pxp}(t, C) = 0$$
$$\forall t \in \mathbb{N}^+, C \in \mathcal{C} : att_{pip}(t, C) = 0$$

The root PEP attests the PDP and PXP at $t = 1$, while both PEP and PDP attest the PIP at $t = 2$. Since in this example all conducted attestations are successful, the attestation schedules never evaluate to -1.

$$att_{pep}(t, C) = \begin{cases} 1, \ t = 1 \wedge C = \{pdp, pxp\} \\ 1, \ t = 2 \wedge C = \{pip\} \\ 0, \ \text{else} \end{cases}$$

$$att_{pdp}(t, C) = \begin{cases} 1, \ t = 2 \wedge C = \{pip\} \\ 0, \ \text{else} \end{cases}$$

Furthermore we consider both attestation and locality trust gains for this example calculation. Table 4.1 shows the development of the trust level for the three paths.

t	$TL^{(pep,pdp,pxp)}$	$TL^{(pep,pdp,pip)}$	$TL^{(pep,pip)}$
0	0	0	0
1	$\eta^{att}(0) * \eta^{loc}(0)$	0	0
2	$\eta^{att}(1) * \eta^{loc}(1)$	$\eta^{att}(1) * \eta^{att}(0)$	$\eta^{att}(0)$

Table 4.1: Development of trust levels over time.

At $t = 0$, no attestations have been conducted yet, so the trust level for all paths is 0. At $t = 1$, the PEP conducts a remote attestation of the attestation container $\{pdp, pxp\}$. This results in an attestation trust gain of $\eta^{att}(0)$ for the transition $pep \rightarrow pdp$ and a locality trust gain of $\eta^{loc}(0)$ for the transition $pdp \rightarrow pxp$. At $t = 2$, both the PEP and the PDP conduct a remote attestation of the attestation container $\{pip\}$. Then the transition $pep \rightarrow pip$ is directly attested with an attestation trust gain of $\eta^{att}(0)$. However, the transition $pep \rightarrow pdp$ now has an attestation trust gain of $\eta^{att}(1)$, since the relevant attestation is one time step in the past. For the same reason the transition $pdp \rightarrow pxp$ now has a locality trust gain of $\eta^{loc}(1)$.

If we assume the dampening factors of the trust gains to be

$$\eta^{att}(t) := \exp(-\frac{1}{10}t)$$

and $\eta^{loc}(t) := \exp(-\frac{1}{15}t)$, the trust level of the entire activity H at time $t = 2$ results to

$$\begin{aligned} \mathit{TrustLevel}^H(2) &= \min\big(\eta^{att}(1) * \eta^{loc}(1), \eta^{att}(1) * \eta^{att}(0), \eta^{att}(0)\big) \\ &= \min\big(\eta^{att}(1) * \eta^{loc}(1), \eta^{att}(1) * 1, 1\big) \\ &= \eta^{att}(1) * \eta^{loc}(1) \\ &= 0.846 \end{aligned}$$

5 Conclusion

In this work we developed a formal model for quantifying trust in distributed usage control systems. After defining the relevant trust dependencies and interacting agents, we developed binary and non-binary trust metrics that quantify the level of trust reached in a certain scenario. While successful attestations positively influence the trust, failed attestations and time progression reduce the reached overall trust level. Finally we showed an example calculation based on the real distributed usage control system that is deployed in the International Data Space.

Possible future work includes investigating how Dempster-Shafer theory [8] could be applied to the formal model. With Dempster-Shafer it is possible to model unawareness and uncertainty of knowledge. It is also helpful in combining degrees of belief from different sources, which makes it promising for representing trust in distributed systems. There already are reputation systems based on Dempster-Shafer theory [12].

Another important approach is to evaluate to what extent the assumptions made by the formal model hold in practice. The presented trust metric is only meaningful if the used remote attestation protocol guarantees integrity verification and secure communication across the distributed system. However, especially for the widespread TPMs this assumption does not hold in all scenarios [11]. A more subtle problem that occurs in practice is the availability of UC components.

Even if the used remote attestation protocol is secure, one can never prevent a malicious operator to deliberately sever communications between local and remote usage control components. In this case it is important that the roots of all affected UC activities are notified about the loss of communication, otherwise the security of the usage control system may be compromised. Even though the formal model cannot directly monitor this, being able to identify relevant UC activities and their trust dependencies is a substantial help in auditing distributed usage control systems for these weaknesses.

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] Anne Anderson et al. “extensible access control markup language (xacml) version 1.0”. In: *OASIS* (2003).

[3] Victor Costan and Srinivas Devadas. “Intel SGX Explained”. In: *IACR Cryptology Archive* (2016), p. 86.

[4] Audun Josang and Roslan Ismail. “The beta reputation system”. In: *Proceedings of the 15th bled electronic commerce conference*. Vol. 5. 2002, pp. 2502–2511.

[5] Stephen Paul Marsh. “Formalising trust as a computational concept”. In: (1994).

[6] Boris Otto et al. *IDS Reference Architecture Model*. Tech. rep. International Data Spaces Association, 2018.

[7] 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.

[8] Glenn Shafer. *A mathematical theory of evidence*. Vol. 42. Princeton university press, 1976.

[9] TCG. “Architecture overview”. In: *Specification Revision* 1 (2007).

[10] H Vagts, T Cosar, and J Beyerer. “Establishing trust in decentralized smart sensor networks”. In: *Mobile Multimedia/Image Processing, Security, and Applications 2011*. Vol. 8063. International Society for Optics and Photonics. 2011, p. 806306.

[11] Paul Georg Wagner, Pascal Birnstill, and Jürgen Beyerer. “Challenges of Using Trusted Computing for Collaborative Data Processing”. In: *International Workshop on Security and Trust Management*. Springer. 2019, pp. 107–123.

[12] Bin Yu and Munindar P Singh. “An evidential model of distributed reputation management”. In: *Proceedings of the first international joint conference on Autonomous Agents and Multiagent Systems: Part 1*. ACM. 2002, pp. 294–301.