Securing Information Gateways with Derivation-Constrained Access Control

Urs Hengartner[1] and Peter Steenkiste[2]

[1] University of Waterloo
uhengart@cs.uwaterloo.ca
[2] Carnegie Mellon University
prs@cs.cmu.edu

Abstract. In pervasive computing environments, information gateways derive specific information, such as a person's location, from raw data provided by a service, such as a videostream offered by a camera. Here, access control to confidential raw data provided by a service becomes difficult when a client does not have access rights to this data. For example, a client might have access to a person's location information, but not to the videostream from which a gateway derives this information. Simply granting access rights to a gateway will allow an intruder into the gateway to access any raw data that the gateway can access. We present the concept of derivation-constrained access control, which requires a gateway to prove to a service that the gateway needs requested raw data to answer a client's authorized request for derived information. Therefore, an intruder into the gateway will be limited in its capabilities. We provide a formal framework for derivation-constrained access control based on Lampson et al.'s "speaks-for" relationship. We demonstrate feasibility of our design with a sample implementation and a performance evaluation.

1 Introduction

Pervasive computing environments are full of services (e.g., sensors, databases, filesystems) that provide raw data. However, clients often do not access this raw data directly, instead they access it indirectly via an *information gateway* that extracts specific information from the raw data. For example, information gateways can derive a user's location from a videostream delivered by a camera or a user's current activity from her calendar. In short, a gateway accesses data offered by a service on behalf of a client.

Access control ensures that only authorized clients are granted access to confidential data. If clients access data via gateways, access control becomes difficult. On the one hand, a gateway needs access rights to the data in order to retrieve this data from a service. On the other hand, granting access rights to a gateway makes it vulnerable in case of an intrusion. Namely, an intruder into the gateway has access to any data that the gateway is authorized to get, and the intruder can actively issue requests for this data to services. Services will answer these requests if the gateway is authorized to access the requested data.

In this paper, we propose *derivation-constrained access control*. Here, before returning data to a gateway, a service requires the gateway to prove that the gateway is

J.A. Clark et al. (Eds.): SPC 2006, LNCS 3934, pp. 181–195, 2006.

Fig. 1. Example gateway

asking for this data in order to answer a client's authorized request for information derived from this data. If there is no such request, an intruder into the gateway will not be able to prove that the gateway is asking for the data on a client's behalf. Therefore, the service will not grant the gateway (and the intruder) access to the requested data.

Let us illustrate derivation-constrained access control with an example application, which we will use throughout the paper. There is a service that locates people and a service that locates devices (see Figure 1). The latter service locates people indirectly by locating their devices, that is, the people location service acts as a gateway and contacts the device location service upon a client's request for a person's location. Assume that Bob asks the people location service for Alice's location information. The people location service then asks the device location service for the location of Alice's laptop. Derivation-constrained access control now makes the people location service prove to the device location service that the people location service is issuing this request in order to answer Bob's authorized request for Alice's location and that the location of Alice's laptop should be used for deriving Alice's location.

The main contribution of this paper is the concept of derivation-constrained access control. Three additional contributions are a formal model of this concept, a sample implementation, and a performance evaluation.

The rest of this paper is organized as follows: We present background material in Section 2. In Section 3, we discuss the concept of derivation-constrained access control in more detail. We introduce our formal model in Section 4 and apply it to an example scenario in Section 5. In Section 6, we present a security analysis. We discuss our sample implementation and its performance in Section 7 and 8, respectively. We comment on related work in Section 9.

2 Background

In this section, we discuss two types of gateways and contrast approaches chosen in related work with derivation-constrained access control. We also give our threat model.

The first type of gateways has the property that an authorized client has access rights both to the information offered by a gateway and to the data provided by the service that the gateway accesses on behalf of the client. For example, Alice has access both to her raw medical data, as collected by individual services (e.g., sensors), and to her health status, as derived by a gateway. In another example, she can access both her calendar data, as provided by a calendar application, and her location information, as derived from her calendar data by a gateway.

Lots of related work deals with this kind of gateways [1, 2, 3, 4, 5, 6]. In the proposed solutions, a client delegates its access right to a gateway, which enables the gateway to access the data at the service. To reduce the influence of an intruder into the gateway,

related work suggests two approaches: First, instead of delegating all its access rights to a gateway, a client should delegate only the subset required by the gateway [3]. Second, a client should delegate only short-lived access rights [1, 2, 4]. Since the proposed solutions rely on the delegation of access rights, we use the term *delegation gateways* for referring to this kind of gateways.

In this paper, we focus on a second type of gateways, which has the property that an authorized client has access rights to the information offered by a gateway, but the client does not have access rights to the data offered by the service that the gateway accesses on behalf of the client. For example, a face detection application acts as a gateway and extracts somebody's location from a videostream. Even though Alice's location can be derived from a videostream and she can access her location information, she cannot access the videostream directly. In another example, Carol might have access to Bob's location information, but she does not have access to Bob's calendar data, from which a gateway derives his location. Finally, a gateway can derive the location of a person from the location of her laptop. A company might decide that, for administrative reasons, its employees should not have access rights to their laptop's location information. On the other hand, an employee should have access rights to her location information. We use the term *derivation gateways* for referring to this kind of gateways.

Related work has largely ignored derivation gateways. Delegation of access rights is not applicable here, since a client does not have access rights to the data offered by a service. Instead, derivation gateways are typically assumed to have separate access rights to data provided by a service. Therefore, access control ignores that this data is used by a gateway for deriving specific information, which makes the gateway vulnerable in case of an intrusion.

Derivation-constrained access control exploits such derivation properties. Namely, it makes a gateway prove to a service that the gateway is asking for data in order to answer a client's authorized request for derived information. The concept has the additional benefit that a gateway does not have to make access decisions. Only a service needs to make such decisions. Therefore, we can run access control in an end-to-end way.

We assume the following threat model: Gateways are not malicious to start with, but they become corrupted if an attacker breaks into them. In the case of corruption, an attacker has full control over a gateway and uses the gateway to contact other services. Corrupted gateways and clients that are authorized to get information offered by a gateway do not collude. Similarly, services and corrupted gateways do not collude.

3 Derivation-Constrained Access Control

In the rest of this paper, we use the term "information" for referring both to (raw) data, as offered by a service, and to derived information, as provided by a gateway.

Derivation-constrained access control requires two basic components. First, we need *derivation-constrained access rights*, which constrain access rights to an entity such that the entity has to prove that it is asking for the information listed in the access right in order to answer an authorized request for derived information. Second, in order to be able to identify derived information, we need formal *derivation properties* between information a gateway asks for and information a client asks for.

Instead of having two separate components, it is possible to directly include derivation properties in access rights. This approach has the drawback that it increases the number of access rights that need to be issued. However, it turns out that the approach restricts the flexibility of intruders that have broken into a gateway. We discuss this tradeoff in Section 6.

We now examine access rights and derivation properties in more detail and describe their application in derivation-constrained access control.

3.1 Derivation-Constrained Access Rights

Access rights to information are specified as follows:

[**Issuer** gives **subject** access to **information** under some **constraints**].

Issuers grant access by defining an access right.
Subjects are granted access rights. A subject can become an issuer by defining another access right and by granting the access right to another subject.[1] Therefore, there can be an entire chain of access rights delegating an access right. We call the first access right in the chain the *initial* access right.
Information describes information (e.g., a location or a videostream) about or from a specific entity (e.g., Alice or a camera in a building). For each type of information, there must be an *owner* who is allowed to define its initial access right. For example, Alice defines the initial access right for her location information and a building administrator defines the initial access right for a videostream provided by a camera in the building.
Constraints define under what conditions the access right is valid. There can be different types of constraints (e.g., time-based). In derivation-constrained access control, a constraint requires that the information in the access right is used for answering a request for derived information. We call access rights having such a constraint *derivation-constrained access rights*.

In our example application, Alice's company, as the owner of Alice's laptop, defines a derivation-constrained access right stating that the people location service can access the location of Alice's laptop.

3.2 Derivation Properties

We need derivation properties to tie information asked for by a gateway to information asked for by a client.

[**Issuer** says that **subject information** is derived from **issuer information**].

Issuers state derivation properties.
Issuer information is the information offered by a service. A gateway uses the issuer information to derive the subject information.
Subject information is the information offered by a gateway. A gateway derives the subject information from the issuer information.

[1] If we did not allow subjects to become issuers, they could proxy on behalf of other entities.

Only the owner of the issuer information and people having access to this information are allowed to specify a valid derivation property for this information.[2] The reason for this requirement is that a derivation property is used for controlling access to the issuer information. If a random client was able to define derivation properties, the client could define a derivation property with information owned by the client as subject information and send an authorized request for this subject information to a gateway. Assuming the gateway is granted access to the issuer information in a (derivation-constrained) access right, the gateway could then successfully query a service for this issuer information and leak the (derived) information to the client.

There is no need to specify derivation properties for each request issued by a client, they need to be defined only when there is a change in the derivation properties of information. Furthermore, the same issuer information can show up in multiple derivation properties. For example, Alice's calendar could be used for deriving both Alice's current location and her current activity. Similarly, subject information can occur in multiple derivation properties. For instance, Alice's location could be derived from Alice's calendar or from her laptop's location information.

Let us revisit our example application. Alice's company, as the owner of Alice's laptop, defines a derivation property that has the laptop's location information as issuer information and Alice's location information as subject information.

3.3 Access Control

Upon receiving a request for information, a service needs to make its access decision based on either access rights that are not derivation-constrained or derivation-constrained access rights and derivation properties. For both cases, the service needs to validate the access rights that grant access to the requesting gateway. Namely, the service must ensure that the information listed in the access right (or in a chain of access rights) corresponds to the information provided by the service. In particular, the access right and the service must agree on the owner of the information. For example, the administrator of a location service run by a company might decide that the company owns employee Alice's location information, as offered by this service. Therefore, the location service must not use access rights which state that Alice owns the information listed in the access right. In this case, the company is the issuer of the initial access right. On the other hand, the administrator of a location service in a mall can declare that Alice owns her location information, as provided by this service. Therefore, the service exploits access rights that list Alice as the owner of her location information. In this case, Alice is the issuer of the initial access right.

If the service makes an access decision based on derivation-constrained access rights and derivation properties, it will also have to validate the following properties:

- There must be a derivation property with the requested information as issuer information.
- There must be a request from a client to the gateway asking for the subject information in the derivation property.

[2] If we prevented people having access from issuing derivation properties, they could proxy.

- There must be a or a chain of access rights that grant the client access to this subject information.
- To avoid replay attacks, the client's request needs to be fresh.

In our example application, the device location service needs both Alice's derivation-constrained access right and the corresponding derivation property. In addition, it requires a fresh request from, for example, Bob asking for Alice's location information and an access right granting Bob access to this information.

4 Formal Model

In this section, we introduce a formal model for defining derivation-constrained access rights and derivation properties. The model allows us to formally reason about access control. It also provides the foundation for our implementation, which is based on digital certificates (Section 7). Our formal model requires a scheme for representing information in access rights and derivation properties. As explained in Section 3, for each type of information, there must be an owner. To simplify associating information with its owner, we make the owner part of the information representation scheme. Namely, we use $A.x$ for referring to information x with owner A. Example information is Alice.location_of_Alice or Building_Administrator.videostream_of_camera.

4.1 Conventional Access Control

We now revisit access control in distributed environments, as discussed by Lampson et al. [4] (and extended by Howell and Kotz [3]), and enhance their formal model to express the exact requirements that a client needs to fulfill in order to be granted access to information. In Section 4.2, this enhancement will allow us to easily add support for derivation-constrained access control to the existing access control model.

For expressing access rights to information, we exploit Howell and Kotz's "restricted speaks-for" relationship, which is based on Lampson et al.'s "speaks-for" relationship. We present the relationship based on an example. The relationship $B \stackrel{T}{\Longrightarrow} A$ denotes that B speaks for A regarding the statements in set T, that is, if A issues a statement that is in T, B also issues this statement. We are interested in statements expressing read access to information, for example, "read $D.x$". In the rest of this paper, we use the shortcut $B \stackrel{D.x}{\Longrightarrow} A$ instead of $B \stackrel{\{\text{"read } D.x\text{"}\}}{\Longrightarrow} A$. As we will see below, this speaks-for relationship grants B access to information $D.x$, assuming that A itself has access to this information.

According to the "handoff axiom" [4], the relationship $B \stackrel{D.x}{\Longrightarrow} A$ can be established by A. Formally, ($\supset$ denotes implication.)

$$\vdash (A \textbf{ says } (B \stackrel{D.x}{\Longrightarrow} A)) \supset (B \stackrel{D.x}{\Longrightarrow} A). \tag{1}$$

A statement of the form A **says** $(B \stackrel{D.x}{\Longrightarrow} A)$ corresponds to an access right, as introduced in Section 3.1, except that there is no support for constraints. For example, Alice **says** $(\text{Bob} \stackrel{\text{Alice.location\_of\_Alice}}{\Longrightarrow} \text{Alice}$ denotes that Alice grants Bob access to

her location information. The speaks-for relationship is transitive, which allows delegation of access rights.

When a service receives a request "read $D.x$" from principal C, it needs to ensure that C speaks for a principal on the service's access control list ("ACL") for this particular resource. Abadi et al. [7] introduce the construct D **controls** s to encode that principal D is on a service's ACL for resource s. The construct is defined as

$$D \textbf{ controls } s \equiv ((D \textbf{ says } s) \supset s).$$

As mentioned in Section 3, the owner D of information $D.x$ is responsible for defining the initial access right to $D.x$, formally, D **controls** "read $D.x$".

We can now summarize access control: Given D **controls** "read $D.x$" (i.e., a service's ACL), $C \stackrel{D.x}{\Longrightarrow} D$ (i.e., a speaks-for relationship derived from a or a chain of access rights), and C **says** "read $D.x$" (i.e., a request), a service concludes "read $D.x$" and grants C access to $D.x$.

For our purposes, the definition of the **controls** construct is not sufficient. It is important to guarantee freshness of a request in order to avoid replay attacks, as stated in Section 3.3. Strictly speaking, freshness demands that a request is new. We relax this requirement and demand only that a request is recent. As it turns out, Lampson et al.'s formal model easily supports this relaxed requirement, and the original requirement is difficult to achieve in our problem setting (see Section 7.2).

To formalize freshness, we observe that Lampson et al. suggest a statement form s **until** t to denote the lifetime t of statement s. We consider a statement fresh if its lifetime has not expired and extend Abadi et al.'s definition of ACLs to support freshness of a request: (**time** is a constant denoting the current time.)

$$D \textbf{ controls } s \equiv (((D \textbf{ says } s) \textbf{ until } t) \wedge (t \geq \textbf{time}) \supset s).$$

4.2 Derivation-Constrained Access Control

We are now ready to formalize derivation-constrained access control, as discussed in Section 3, by extending the definition of D **controls** s .

We first introduce the convention to mark information in derivation-constrained access rights with the + sign (e.g., A **says** $(B \stackrel{D.x^+}{\Longrightarrow} A)$). This way, when a service detects marked information in a speaks-for relationship, it uses this relationship only for derivation-constrained access control, not for conventional access control.

We also introduce a notation for expressing derivation properties. The statement

$$E.y \mapsto D.x \tag{2}$$

denotes that subject information $E.y$ can be derived from issuer information $D.x$.

As explained in Section 3.2, we assume that for a derivation property to hold, the principal owning the issuer information or a principal speaking for the owner needs to issue the property, that is,

$$\vdash (F \textbf{ says } (E.y \mapsto D.x)) \wedge (F \stackrel{D.x}{\Longrightarrow} D) \supset (E.y \mapsto D.x). \tag{3}$$

We can now formulate derivation-constrained access control. In particular, we have a service use a different definition for D **controls** s if s involves information marked with the + sign. Formally,

$$
\begin{aligned}
& D \textbf{ controls } \text{“read } D.x^{+}\text{”} \qquad (4)\\
& \equiv (((D \textbf{ says } \text{“read } D.x^{+}\text{”}) \textbf{ until } t_1) \wedge (t_1 \geq \textbf{time})\\
& \wedge (E.y \mapsto D.x)\\
& \wedge ((E \textbf{ says } \text{“read } E.y\text{”}) \textbf{ until } t_2) \wedge (t_2 \geq \textbf{time})\\
& \supset \text{“read } D.x^{+}\text{”}).
\end{aligned}
$$

The first condition is straightforward: There must be a fresh and authorized request, “read $D.x^{+}$”. In the second condition, we require that there is information $E.y$ that can be derived from $D.x$.[3] The third condition calls for a fresh and authorized request for the derived information, $E.y$.

Note that the third condition can be replaced by a condition requiring a request for $E.y^{+}$ and conditions requiring the existence of another derivation property and of another request. This property makes deployment of our mechanism feasible in scenarios where a client's request for information is dealt with by multiple, cascaded gateways.

5 Example Application

In this section, we demonstrate a sample application of derivation-constrained access control. We assume that individuals and gateways or services each own a public and a private key. We use the public key for identifying entities. For example, the notation K_{Alice} denotes Alice's public key. Entities use the private key for signing statements.

Assume that a people location service, PL, derives Alice's location information, $K_{\mathrm{Alice}}.\mathrm{loc\_Alice}$, from her laptop's location information, $K_{\mathrm{ACME}}.\mathrm{loc\_Alice\_laptop}$, where the laptop is owned by Alice's company, ACME, and its location information is offered by a device location service. Note that the two types of information are owned by two different entities. We show the relevant statements in Table 1.[4] We start with statements that individuals make before an actual request is issued. Alice grants Bob access to her location information (S1). We expect this statement to be long lived. ACME grants the people location service access to Alice's laptop's location information in a derivation-constrained access right (S2), again in a long-lived statement. ACME also states that the laptop's location information can be used for deriving Alice's location information (S3). Finally, the device location service has ACME on its ACL for the laptop's location information (S4).

Bob sends a request for Alice's location information to the people location service (S5) and issues a short-lived access right for Alice's location information to the people location service quoting (denoted by “|”) Bob (S6). This way, the people location service must quote Bob when it wants to use this access right. As observed by

[3] Derivation properties can also have lifetimes; we leave them away for readability reasons.

[4] In a complete approach, we would also have to take communication channels used by principals into account, but we ignore them for simplicity reasons. Similarly, we assume that the axioms introduced by Lampson et al. [4] have been modified to take lifetimes into account.

Table 1. Statements in the people location scenario. (S1)-(S4) are established during setup, (S5)-(S16) are established during access control.

Step	Statements
(S1)	$(K_{\text{Alice}}$ **says** $K_{\text{Bob}} \xRightarrow{K_{\text{Alice}}.\text{loc\_Alice}} K_{\text{Alice}})$ **until** t_1
(S2)	$(K_{\text{ACME}}$ **says** $K_{\text{PL}} \xRightarrow{K_{\text{ACME}}.\text{loc\_Alice\_laptop}^{+}} K_{\text{ACME}})$ **until** t_2
(S3)	K_{ACME} **says** $K_{\text{Alice}}.\text{loc\_Alice} \mapsto K_{\text{ACME}}.\text{loc\_Alice\_laptop}$
(S4)	K_{ACME} **controls** $K_{\text{ACME}}.\text{loc\_Alice\_laptop}^{+}$
(S5)	K_{Bob} **says** "read $K_{\text{Alice}}.\text{loc\_Alice}$"
(S6)	$(K_{\text{Bob}}$ **says** $K_{\text{PL}}\vert K_{\text{Bob}} \xRightarrow{K_{\text{Alice}}.\text{loc\_Alice}} K_{\text{Bob}})$ **until** t_3
(S7)	$K_{\text{PL}}\vert K_{\text{Bob}}$ **says** "read $K_{\text{Alice}}.\text{loc\_Alice}$"
(S8)	K_{PL} **says** "read $K_{\text{ACME}}.\text{loc\_Alice\_laptop}^{+}$"
(S9)	$(K_{\text{PL}} \xRightarrow{K_{\text{ACME}}.\text{loc\_Alice\_laptop}^{+}} K_{\text{ACME}})$ **until** t_2
(S10)	$(K_{\text{ACME}}$ **says** "read $K_{\text{ACME}}.\text{loc\_Alice\_laptop}^{+}$") **until** t_2
(S11)	$K_{\text{Alice}}.\text{loc\_Alice} \mapsto K_{\text{ACME}}.\text{loc\_Alice\_laptop}$
(S12)	$(K_{\text{PL}}\vert K_{\text{Bob}} \xRightarrow{K_{\text{Alice}}.\text{loc\_Alice}} K_{\text{Bob}})$ **until** t_3
(S13)	$(K_{\text{Bob}}$ **says** "read $K_{\text{Alice}}.\text{loc\_Alice}$") **until** t_3
(S14)	$(K_{\text{Bob}} \xRightarrow{K_{\text{Alice}}.\text{loc\_Alice}} K_{\text{Alice}})$ **until** t_1
(S15)	$(K_{\text{Alice}}$ **says** "read $K_{\text{Alice}}.\text{loc\_Alice}$") **until** t_3
(S16)	"read $K_{\text{ACME}}.\text{loc\_Alice\_laptop}^{+}$"

Howell and Kotz [8], this quoting prevents a gateway, like the people location service, from having to make an access decision itself. Instead, by quoting a client, the gateway notifies a service of the identity of the client, which allows the service to make an access decision in an end-to-end way. Statement (S7) shows an example of quoting: The people location service forwards Bob's request to the device location service by quoting Bob. The people location service also issues a request for the location of Alice's laptop to the device location service (S8).

The device location service validates the authenticity of Statement (S2) using Axiom (1) and deduces that the people location service can speak for ACME on behalf of Alice's laptop (S9). Therefore, ACME supports the request of the people location service (S8) for the laptop's location information (S10). The device location service also validates Statement (S3) using Axiom (3) and deduces that the derivation property is valid (S11). Based on the people location service's access right (S6), the device location service concludes that the people location service (quoting Bob) can speak for Bob (S12) and thereby Bob supports (S13) the request of the people location service (S7). After validating that Bob can speak for Alice (S14) based on his access right (S1), the service concludes that Alice also supports the request (S15). (We assume that $t_3 \ll t_1$ since Statement (S1) is long lived, whereas Statement (S6) is short-lived.) Given the ACL for the laptop's location information (S4) and Definition (4), the device location service uses Statements (S10), (S11), and (S15) to deduce that the people location service should be granted access (S16) (assuming $t_2 \geq \mathbf{time}$ and $t_3 \geq \mathbf{time}$).

Note that among the statements in Table 1, only Statements (S1)-(S6) need to be specified by individuals, all the other statements can be derived or specified

automatically by the access control framework (see Section 7). Furthermore, only two of these statements, (S5) and (S6), need to be issued for each request. (In Section 7.2, we present an optimization that combines these two statements in a single statement.) Statements (S1)-(S4) need to be re-issued only when they expire or when there is a change in the derivation properties of information. Among these four statements, Statement (S3) is the only additional statement that is required by derivation-constrained access control, the other three statements are also required in a traditional access control scenario, where the people location service is granted unrestricted access to the laptop location information. (In addition, this scenario would require a statement denoting the ACL of the people location service because this service would have to run access control.)

6 Security Analysis

Since our formal model is based on Lampson et al.'s model, we inherit its properties. For instance, the model does not prevent a principal having an access right from delegating this access right to other principals. For derivation-constrained access control, this principle implies that an intruder breaking into a gateway can delegate the gateway's (derivation-constrained) access rights to other, potentially corrupted gateways. While this attack will not grant the first gateway access to additional information, the other gateways could exploit the delegated access rights if there was an authorized request from a client to them.

As mentioned in Section 3, we keep access rights and derivation properties separate. A disadvantage of this approach is that intruders into a gateway get more flexibility. Assume that there are multiple derivation properties, all of them having the same issuer information. For example, multiple kinds of information can be derived from someone's calendar. The owner of the calendar grants derivation-constrained access to a gateway because the owner expects the gateway to provide one particular derivation. (For example, the owner expects the gateway to provide location information.) If an authorized client now asks the gateway for subject information listed in any of the derivation properties, the gateway will be granted access, even though the owner does not expect the gateway to perform the derivation listed in the derivation property exploited by the gateway. Therefore, an intruder into a gateway can increase its chances of success by trying to trick authorized clients into contacting the corrupted gateway for any of the different types of subject information. There are different ways to address this attack: We can instruct clients which gateways to use for retrieving a particular kind of information, for example, when a client is granted an access right to this information. We can also avoid the attack by having the owner of the issuer information combine the derivation-constrained access right granted to the gateway and the derivation property targeted at this gateway in a single statement such that an attacker cannot exploit them separately. However, this approach could require the owner of the issuer information to issue more access rights. Namely, if a gateway extracted multiple information items from the issuer information (e.g., multiple people's location could be derived from a camera picture), the gateway will need separate access rights for each information item. When derivation properties are separate, the gateway requires only one access right.

Derivation-constrained access control mainly targets services that provide dynamic information (e.g., location information). An intruder into a gateway gets to see information retrieved by the gateway upon an authorized request from a client. Therefore, if this information was static, an attacker could just wait for an authorized request to occur, and there is less incentive for the intruder to issue its own requests in the meantime.

7 Implementation

Let us discuss our sample implementation of derivation-constrained access control.

7.1 Access Control

When receiving a request asking for access to information $D.x$ or $D.x^+$, a service S needs to establish "read $D.x$" or "read $D.x^+$", based on Definitions (2) or (4). There are multiple ways to establish these statements. A service can retrieve a set of access rights and derivation properties, locate relevant statements in this set, and validate them. However, this approach puts a potentially heavy load on the service. It is really only the validation step that the service has to perform itself, whereas it is possible to offload the first two steps to some other entity. The validation step tends to be cheap (see Section 8), which makes it possible for resource-limited services (e.g., a sensor) to run this step. In our implementation, we make the client and the gateway gather any statements required for being granted access and have them ship these statements, together with the request, to the gateway and the service, respectively. Resource-limited clients can further offload the gathering of statements, if needed. We implemented derivation-constrained access control within an existing framework for such *proof-based* access control [8].

7.2 Statements

To express the access rights and derivation properties introduced in Section 4, we rely on digital certificates. Digital certificates are signed, which makes it easy to identify the issuer of a statement. Our framework is based on SPKI/SDSI certificates [9]. The existing SPKI/SDSI specification supports the inclusion of constraints in an access right. However, interpretation of a constraint is left to an application. Therefore, we extended the specification and let derivation-constrained access rights and derivation properties become first-class citizens. This way, we can support derivation-constrained access control directly in the access control framework.

As shown in Section 5, a client that issues a request also needs to generate a short-lived access right for the information in the request to the gateway. If there are multiple, cascaded gateways, each will have to issue short-lived access rights to the next gateway in the chain. As we will see in Section 8.1, generating a digital signature covering such an access right is expensive. In our implementation, we reduce cost by not making entities issue short-lived access rights. Instead, we have the original issuer of a request sign this request, where the lifetime of this signature is short. Gateways just forward this signed request. We assume that any entity that is in the possession of a signed request is implicitly granted access to the information in the request. Formally, this entity can speak for the original issuer of the request on behalf of the information in the request. To be secure, this optimization requires two precautions: First, we have to ensure that

attackers cannot snoop traffic and obtain access rights by observing signed requests. Therefore, communication has to be confidential. Note that confidentiality is required anyway because we exchange confidential information (such as a person's location information). Second, a service can use a speaks-for relationship that is implicitly derived from a signed request only for running access control for this particular request; it must never reuse such a relationship for other requests. The latter precaution is required since gateways do not run access control. Without it, an attacker could contact a gateway that previously obtained information from a service and have it contact the service again. If the service reused the previously derived speaks-for relationship, it would grant the gateway access again (unless the statement has expired).

To support the **until** construct, we exploit that SPKI/SDSI certificates can have a lifetime associated with them and have the issuer of a statement assign a lifetime to the statement. We use lifetime to achieve freshness. We could also integrate a nonce-based approach, where the entity that wants to issue a request to a target receives a nonce from the target and returns this nonce in its request. However, our communication pattern, where a client interacts only with the gateway, but not with the service, does not allow for nonces. There is no way for the service to give a nonce to the client to prove freshness of the client's request.

7.3 Execution Environment

We use the Aura ubiquitous computing environment [10] as our testbed for the implementation and deployment of derivation-constrained access control. Entities within this environment communicate with each other using a protocol that exchanges XML-encoded messages embedded in HTTP requests/responses. We extended this protocol to support the transmission of proofs of access, as explained in Section 7.1. SSL gives us peer authentication and message confidentiality and integrity.

8 Evaluation

We present a measurement-based analysis of derivation-constrained access control and discuss the results.

8.1 Measurements

We measure the time it takes for a gateway and a service to process a request, where processing includes access control. Our scenario roughly corresponds to the example application from Section 5. We assume that Alice grants Bob access to her location information. Bob contacts a service providing people location information. This service acts as a gateway and locates people by locating their devices. Namely, it contacts a service providing location information about badges worn by people. Alice, as the owner of the badge, specifies a derivation property between her badge's location information and her location information. She also issues a derivation-constrained access access right granting the people location service access to the badge's location information.

Our experiments run on a Pentium IV/2.5 GHz with 1.5 GB of memory, Linux 2.4.20, and Java 1.4.2. The public and private key operations operate on RSA keys with a size of 1024 bit. All experiments are run 100 times.

Table 2. Mean and standard deviation of elapsed time for security-related operations (in bold) and other expensive operations [ms]

Entity	Step	μ (σ)
Client	**Request generation**	27 (3)
Client	**SSL sockets creation**	52 (4)
Gateway	**Access control**	2 (1)
Gateway	Gather badge information	26 (3)
Gateway	**Request generation**	41 (3)
Gateway	**SSL sockets creation**	52 (5)
Service	**Access control**	3 (2)
Service	Gather location information	49 (30)
	Total	310 (37)

It takes about 310 ms for the client to send a request to the gateway and to receive a response. We present more detailed results in Table 2. The terms "client", "gateway", and "service" represent Bob, the people location service, and the badge location service, respectively. The most expensive operations are the two SSL sockets creations, one pair for the client-gateway connection and another pair for the gateway-service connection. For opening an SSL connection, the two peers need to authenticate each other. Each authentication requires an RSA signing or decryption operation operation, which takes about 17 ms. RSA is also responsible for most of the cost of generating a request, because a request needs to be signed. In our current implementation, any entity issuing a request needs to sign it. This approach supports scenarios, where there are multiple, cascaded gateways. If an entity knew that it is directly contacting a service, it could omit this signature. For example, the people location service would not have to sign the request for the badge's location information.

Compared to the cost for creating SSL sockets and generating a request, access control is cheap. The main cost is signature verification, which takes about 1 ms per signature. The gateway validates the signatures of the client's request and of the access right granting access to the client. Strictly speaking, derivation-constrained access control does not require a gateway to make an access decision. However, we have the gateway make such decisions, since they are cheap and can help against denial-of-service attacks, where clients send unauthorized requests to a gateway. The service needs to make an access decision. It validates the signatures of the client's request, of the access right granting access to the client, of the access right granting derivation-constrained access to the gateway, and of the derivation property. Overall, it has to validate four signatures.

Let us compare the cost of derivation-constrained access control to the cost of traditional access control, where the owner of the information provided by a service issues a long-lived access right to a gateway. This approach is less expensive than our approach. There is no need for the client to sign its request to the gateway (i.e., issue a short-lived access right to the gateway) and for the service to validate this access right and the access right of the client. However, long-lived access rights make the gateway vulnerable in case of an intrusion. Furthermore, derivation-constrained access control does not require a gateway to make an access decision, whereas the alternative approach does.

9 Related Work

Lots of related work deals with delegation gateways [1, 2, 3, 4, 5, 6]. We have discussed the limitations of these schemes in Section 2 and provide a more detailed discussion of this work in the first author's Ph.D. thesis [11, Chapter 4].

Derivation-constrained access control is based on the concept of rights amplification (also called privilege escalation). This concept has been used in other access control models. In Bertino et al.'s model [12], an access right becomes valid whenever an access right that it depends on becomes valid. In our scenario, the former access right could be a gateway's access right to information offered by a service and the latter one a client's access right to information provided by the gateway. However, the presented model applies only to a centralized environment, in which there is an administrator that manages both access rights and dependencies between access rights. In pervasive computing, individuals issue access rights and establish derivation properties. Some operating systems based on capabilities (e.g., Hydra [13]) also support rights amplification. Again, these access control models target a centralized environment. Jajodia et al. [14] present several possibilities for deriving access rights, but their model misses a temporal component and applies only to centralized environments.

In derivation-constrained access control, an intruder into a gateway gets to see any information given to the gateway when a service answers an authorized request from the gateway. We could avoid this information leak by having the service encrypt information such that only the client can decrypt it and by having the gateway compute on the encrypted information. This approach obviously worked if the gateway did not need to perform any computations on the information (e.g., when deriving the location of a person from the location of her laptop). The approach is also applicable to some very specific scenarios, where the gateway does have to run some computations on the encrypted information [15]. However, it is unclear how practical this approach is in general.

To be sure that our framework performs correctly, we need to prove that the access control logic is sound by incorporating our extensions into Abadi et al.'s semantic model [7]. (Alternatively, we could prove all our axioms as theorems in a logic known to be sound (e.g., Appel and Felten's logic [16]).) This is ongoing work.

10 Conclusions and Future Work

We introduced the concept of derivation-constrained access control, which allows one to limit the capabilities of an intruder into a gateway. We presented a notation for expressing derivation-constrained access rights to information and derivation properties between information, and we proposed a mechanism for exploiting both of them in derivation-constrained access control. The sample implementation and its deployment demonstrate the feasibility of our design. The performance of access control is competitive. A large fraction of the cost is caused by expensive, but required operations for authenticating peers.

We are deploying derivation-constrained access control in additional services, which provide other information than location information (e.g., task information).

Acknowledgments

We thank the anonymous reviewers for their comments. This research was supported by the Army Research Office through grant number DAAD19-02-1-0389 and by the NSF under award number CNS-0411116.

References

1. Gasser, M., McDermott, E.: An Architecture for Practical Delegation in a Distributed System. In: Proceedings of IEEE Symposium on Security and Privacy. (1990) 20–30
2. Kornievskaia, O., Honeyman, P., Doster, B., Coffman, K.: Kerberized Credential Translation: A Solution to Web Access Control. In: Proceedings of 10th Usenix Security Symposium. (2001)
3. Howell, J., Kotz, D.: A Formal Semantics for SPKI. In: Proceedings of 6th European Symposium on Research in Computer Security (ESORICS 2000). (2000) 140–158
4. Lampson, B., Abadi, M., Burrows, M., Wobber, E.: Authentication in Distributed Systems: Theory and Practice. ACM Transactions on Computer Systems **10**(4) (1992) 263–310
5. Neuman, B.: Proxy-Based Authorization and Accounting for Distributed Systems. In: Proceedings of International Conference on Distributed Computing Systems. (1993) 283–291
6. Sollins, K.R.: Cascaded Authentication. In: Proceedings of IEEE Symposium on Security and Privacy. (1988) 156–163
7. Abadi, M., Burrows, M., Lampson, B.: A Calculus for Access Control in Distributed Systems. ACM Transactions on Programming Languages and Systems **15**(4) (1993) 706–734
8. Howell, J., Kotz, D.: End-to-end authorization. In: Proceedings of 4th Symposium on Operating System Design & Implementation (OSDI 2000). (2000) 151–164
9. Ellison, C., Frantz, B., Lampson, B., Rivest, R., Thomas, B., Ylonen, T.: SPKI Certificate Theory. RFC 2693 (1999)
10. Garlan, D., Siewiorek, D., Smailagic, A., Steenkiste, P.: Project Aura: Towards Distraction-Free Pervasive Computing. IEEE Pervasive Computing **1**(2) (2002) 22–31
11. Hengartner, U.: Access Control to Information in Pervasive Computing Environments. PhD thesis, Computer Science Department, Carnegie Mellon University (2005) Available as Technical Report CMU-CS-05-160.
12. Bertino, E., Bettini, C., Samarati, P.: A Temporal Authorization Model. In: Proceedings of 2nd ACM Conference on Computer and Communications Security (CCS 1994). (1994) 126–135
13. Cohen, E., Jefferson, D.: Protection in the Hydra Operating System. In: Proceedings of 5th ACM Symposium on Operating Systems Principles. (1975) 141–160
14. Jajodia, S., Samarati, P., Sapino, M.L., Subrahmaninan, V.S.: Flexible Support for Multiple Access Control Policies. ACM Transactions on Database Systems **26**(2) (2001) 214–260
15. Song, D., Wagner, D., Perrig, A.: Practical Techniques for Searches on Encrypted Data. In: Proceedings of 2000 IEEE Symposium on Security and Privacy. (2000)
16. Appel, A.W., Felten, E.W.: Proof-Carrying Authentication. In: Proceedings of 6th ACM Conference on Computer and Communications Security. (1999)