Policy Provisioning for Distributed Identity Management Systems

Hidehito Gomi

Yahoo! JAPAN Research, 9-7-1 Akasaka, Minato-ku, Tokyo 107-6211, Japan
hgomi@yahoo-corp.jp

Abstract. A policy provisioning framework is described that supports the management of the lifecycle of identity information distributed beyond security domains. A model for creating data handling policies reflecting the intentions of its system administrator and the privacy preferences of the data owner is explained. Also, algorithms for systematically integrating data handling policies from system entities in different administrative domains are presented. This framework enables data handling policies to be properly deployed and enforced in a way that enhances security and privacy.

1 Introduction

Many applications are being executed in distributed systems and among multiple different organizations with the ongoing development of the Internet. Personal information in such environments is often exchanged beyond the boundaries of security domains. It is generally difficult for both administrators and owners to control data once they are propagated outside their security domains. Thus, identity management in distributed environments is one of the most important issues in preserving security and privacy.

There have been several technical projects on identity management including access control and privacy management. Privacy-aware access control [1, 2] has especially aimed at incorporating privacy-related policies into traditional access control policies. Another emerging concept of *identity governance* [3] has addressed user-centric control of access and introduced a method of tracking data for propagating identity information. Although these research efforts have enabled fine-grained access control for managing identity from security and privacy perspectives, they have not fully addressed how data handling policies can be created and integrated, which satisfy the different requirements of distinct actors in different security domains from a practical viewpoint. Since these actors are involved with data practices, but generally have different responsibilities, the enforcing policies need to be created from administrative and privacy standpoints.

This paper proposes a policy provisioning framework that helps to manage the lifecycle of identity information using handling polices that reflect its system administrator's intentions and its data owner's preferences from both administrative and privacy viewpoints. Also described are algorithms that enable data

E. de Leeuw, S. Fischer-Hübner, L. Fritsch (Eds.): IDMAN 2010, IFIP AICT 343, pp. 130–144, 2010.

handling policies or privacy preferences to be created and integrated from multiple actors to control access to identity information. This work focuses on collaboratively building a policy provisioning model and framework for distributed identity management systems, whereas the representation of policies and detailed resolutions on policy conflicts are beyond the scope of this work.

The rest of this paper is organized as follows. Section 2 presents related work and Section 3 introduces a policy provisioning model. Section 4 describes a policy provisioning framework based on the proposed model. Section 5 discusses several issues related to policy management and Section 6 concludes the paper with a summary of the key points and an outline of future work.

2 Related Work

This section highlight research efforts in the area of access control, privacy management, and policy specification languages related to the work presented in this paper.

The idea of controlling access to data even after they have been disseminated has been considered especially by the digital rights management (DRM) community [4, 5]. Their work has focused on protecting digital content from unauthorized copying and distribution by disseminating packages containing the content data and access control policies. Several efforts [6, 7] toward managing privacy have introduced the concept of "sticky policies", in which handling policies are directly associated with personal information. In their approaches, users retain control over their personal information even after it has been disclosed by enforcing its privacy policies, which reflect their preferences about how it is to be used next at its recipient site. The work here inherits the basic idea of tight bundling of data and policy described above, regardless of whether the type of policy is security or privacy related.

There have been numerous research efforts related to extensions of traditional mechanisms for access control to protect privacy [1, 2]. Ardagna et al. [2] proposed a privacy-aware system to control access that enforced access control policies together with privacy policies such as release and data handling policies that regulated the use of personal information in secondary applications. They focused on the introduction of data handling policy language and the integration of traditional access control and data handling policies created from two actors, i.e., a service provider that managed personal information and a user who originally had the information. However, their work did not describe how data handling policies were created and deployed in a system that consisted of entities that had distinct responsibilities or roles and that supported multiple chains to disseminate data among those entities. The work described here instead focuses on policy management in which a data managing provider collaboratively establishes data enforcing policies.

Other relevant work on privacy management has been identity governance, which is an emerging concept to provide fine-grained conditional disclosure of identity information and enforce corresponding data handling policies. Liberty

Alliance specifies fundamental privacy constraints on such governance as the use, display, retention, storage, and propagation of identity information [8]. Gomi [3] introduced privacy-aware tracking policies and a data mechanism for monitoring the status of a user's identity information and enforcing its privacy policies to regulate its secondary use after it had been disseminated. Although theses allow access control that enhances privacy by defining new types of expressive privacy policies, they do not fully take into consideration the integration or composition of polices among distinct actors located in different administrative domains. The proposed framework addresses a method of transmitting and incorporating data handling policies associated with shared identity information between actors.

There has been a great deal of work on description languages and constraints for privacy policies. P3P [9] and its complement APPEL [10] provide the means for expressing comprehensive user preferences. XACML [11] specifies an access control language to describe access control constraints and provides privacy extensions to support privacy related constraints. EPAL [12] provides a privacy policy language for governing data practices in enterprise systems. The Liberty Identity Governance Framework (IGF) [8] specifies privacy constraints. Although the framework proposed here assumes such an expressive privacy policy and access control language as basic building blocks, it focuses on managing the lifecycles within which security and privacy policies are enforced irrespective of their schema or the format they are represented in.

Relevant work has been done in the field of policy management such as policy integration and conflict resolution. Mazzoleni et al. [13] proposed policy integration algorithms for XACML. They believed that XACML had not been built to manage security for systems in which enterprises were dynamically constructed with the collaboration of multiple independent subjects. The approach proposed here is relevant in that entities located in different security domains collaboratively share data and their policies. Belokosztolszki and Moody [14] introduced meta-policies for distributed Role-Based Access Control (RBAC). They found it difficult to specify a policy that would not conflict with local requirements. Their work is relevant in that they considered a hierarchical structure for managing policy in distributed systems. Bettini et al. [15] formalized a rule-based policy framework that controlled access by user requests for action by evaluating rules associated with provisions and obligations, which were pre-conditions to be satisfied before and post-conditions to be satisfied after the action was performed. Their framework provided a mechanism for reasoning about the policy rules in the presence of provisions and obligations in a single administrative domain to derive an appropriate set of these. In contrast, the framework presented here composes a set of policies using different actors with different responsibilities in distinct domains and it enforces the composed policies reflecting a system administrator and a user in distributed identity management systems.

3 Model

This section explains a model for policy provisioning.

A **data subject (DS)** is an individual to whom personal data is related. A DS delegates the secure management and convenient utilization of his or her personal data to other entities specifying privacy preferences on how the data is to be handled by them. A DS can demonstrate his or her wishes by means of *consent* to questions from other entities as one of representations of privacy preferences.

A **data controller (DC)** is an entity that maintains DSs' personal data on his or her behalf in compliance with its own privacy or security policies reflecting both their privacy preferences. A DC can securely provide a DS's personal data to another entity in a different administrative domain on the basis of the agreement with the entity on how the data are to be used and handled. A DC is liable for securely managing and propagating a DS's personal data.

A **data processor (DP)** is an entity that processes a DS's personal data obtained from a DC in conformity with the agreement reached with the latter on how the data are to be handled. A DP is placed in a distinct administrative domain from that of a DC. A DP is liable for handling personal data originally managed by a DC. This liability is different from that for a DC since a DP does not need to maintain or determine the purposes for which the data are processed, and since this liability depends on the agreement on data processing between a DC and a DP.

The DC and DP are not actual entities; they are simply roles in the model. Therefore, a single entity can play both roles. That is, when entity e_1 acting as a DP receives personal data from entity e_2 acting as a DC with an agreed upon policy allowing e_1 to store and further propagate the received data to the other entity, e_3, e_1 can act as a DC for e_3. In other words, the relationship between DC and DP is relative and one-way specific to the pair of two entities for the particular types of personal data in this model. In this example, e_1 can be a DC for e_3, but cannot be a DC for e_2, because e_2 was originally a DC for e_1.

It is assumed that these entities are trusted and can be expected to comply with the agreement. The purpose of this work was to establish an agreement on data handling between trusted entities and create and deploy policies to be appropriately enforced, rather than to detect their misbehaviors.

3.1 Policy Binding to Data

Data and their handling policies in this model are tightly associated. When a DC receives an access request to data, the DC determines whether to grant or deny the access enforcing the handling policies associated with the data. When a DP attempts to process data, the DP also uses the handling policies associated with the data to make an authorization decision on the data processing.

If a DC needs to provide a DP with personal data, the DC encapsulates the data and their associated policies and transfers both to the DP. The DP complies with the policy agreed upon and received from the DC prior to receiving the data. Namely, agreed upon policies migrate with the data to govern the data practices of a DC that receives both the data and policies. The agreed upon policies

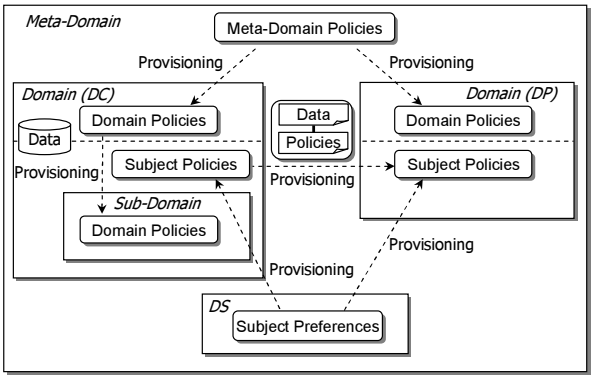


Fig. 1. Hierarchical Policy Model

correspond to an agreement between a DC and a DP when data are transferred and are the legal grounds for appropriately restricting data processing by a DP.

The encapsulation and transport mechanisms of data and policies are beyond the scope of this model. Instead, it focuses on the design of the framework for hierarchically developing policies among distinct entities, which will be described in the next section.

3.2 Policy Hierarchy

Figure 1 outlines a hierarchical policy model that encompasses the defined entities and exchanged policies.

The *Meta-Domain* is a meta-organization or system such as an industrial department or a governmental body to which the *Domains* belong. The Domain is an administrative organization or system independent of others that acts as DC or DP entities. A meta-domain and a domain have a relative association. The *Sub-Domain* belongs to its upper class domain. The relationships between the Meta-Domain and Domain, and Domain and Sub-Domain are hierarchical. These relationships generally hold true without limiting the representation in Fig. 1. A domain acting as a DC provides a DS with a service that manages the DS's personal data. A domain acting as a DP provides a DS with a service that uses a DS's personal data.

A meta-domain has *Meta-Domain Policies* that are meta-level and general policies constraining the activities of all domains that belong to the meta-domain, by reflecting its laws or regulations with which the domains need to comply.

Domain Policies are organizational domain-specific policies that inherit the meta-domain policies in the meta-domain, and are not specific to DSs. When a DC propagates personal data to a DP, the DC and the DP agree on a set of policies on how the data are to be used and handled prior to being propagated. The agreed upon policies migrate with the data from the DC to the DP.

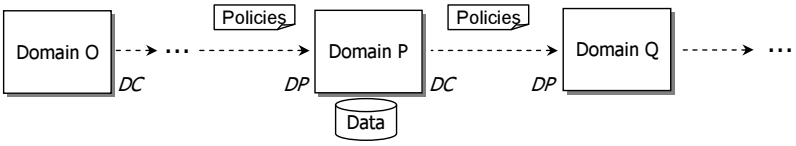


Fig. 2. Policy Provisioning Chain

Subject Policies are specified by a domain for a DS, reflecting *Subject Preferences*, which are a DS's privacy-related preferences for handling the DS's personal data. As a result of incorporating subject preferences into subject policies, a DS needs to follow the subject policies specified by domains to enjoy their services. More detailed descriptions on how these policies are created and provided will be given in the sections that follow.

3.3 Policy Provisioning Chain

A DS's data possibly propagates from domain to domain. Here, the data handling policies associated with the data also propagate from a domain acting as a DC to a domain acting as a DP. The flow of the policies constitutes a chain of domains as seen in Fig. 2.

When domain P manages data, it obtains meta-domain policies from its meta-domain, and additionally agreed upon policies if the data have originally been propagated by another domain acting as a DC to incorporate them into its domain policies. P, acting as a DC, propagates the data and their handling policies to domain Q, acting as a DP, after P and Q have agreed upon the policies. As a result of data being propagated and policies being agreed on, Q becomes responsible for handling the data. In this way, if data propagate from domain to domain, their associated handling policies change to ones reflecting local domain policies and propagate together with data for enforcing the behavior of another domain receiving the data. The first entity that provides policies in a provisioning chain is called the *Root Domain*, which is denoted by domain O in Fig. 2.

There are two types of relationships between adjacent domains in the policy provisioning chain. The first is a hierarchical relationship. In this case, since the upper domain manages the lower domain in a policy hierarchy, the lower domain inherits the policies from those of the upper domain. The second is a propagation relationship in different domains that are placed on the same level in the policy hierarchy. Here, policies are propagated from a DC to a DP after they reach agreement on how data are to be handled.

3.4 Policy Components

This model has several components related to policies, which are relevant to the ones defined in XACML.

- *Action.* An action is a specific activity that invokes a function call or sends a request to other domains.
- *Data.* Data represents a subject's personal data identified by its data type.
- *Rule.* A rule is the basic element of a policy.
- *Policy.* A policy is a combination of one or more rules.
- *Policy set.* A policy set is a collection of one or more policies.

Policies are represented in the following sections in declarative form irrespective of the language or format for coding policies.

4 Policy Provisioning Framework

This section explains detailed procedures on managing the lifecycle of policies as well as on the data to be managed using the policies.

4.1 Policy Creation

Meta-Domain Policy Creation. Meta-domain policies are created by the administrator of the domain. Since a root domain corresponds to an administrative organization such as an industrial company or governmental body, its meta-policies are created from the administrative viewpoints of privacy protection laws or cooperate compliance.

There are two types of meta-domain policies, i.e., *common policies* and *governance policies*. Common policies are general and do not specify concrete constraints in them such as data types, subjects, and context. For example, a meta-domain policy statement in a natural language is "No entities must propagate personal data owned by subjects to other subjects and domains without their consent." This statement can be specified in the following declarative form:

$$\begin{aligned}(\text{P0.a}) : \neg doable(a) \Leftarrow\ & subject(u_1) \wedge subject(u_2) \wedge own(u_1, data) \\ & \wedge\ action(send(u_1, u_2, data), a) \wedge \neg consent(u_1, a).\end{aligned}$$

In the above representation, operators $\neg$, $\wedge$, and $\Leftarrow$ respectively specify logical negation, conjunction, and reverse implication. Axiom $doable(\cdot)$ indicates that a specified action is allowed to be executed if all the constraints specified after operator $\Leftarrow$ are satisfied. $subject(u)$ means that u is a DS in this domain. In $action(send(u_1, u_2, data), a)$, action a is defined as an action where DS u_1 propagates $data$ to DS u_2. Axioms $own(u_1, data)$ and $consent(u_1, a)$ respectively indicate that u_1 owns $data$ and that u_1 consents to the execution of action a.

Governance policies are meta-level and directive ones for managing the privileges and behavior of sub-domains. For example, information on personal attributes such as name and address can ultimately be managed by the DS that owns them, assuming the concept is acceptable from the viewpoint of local laws.

$$\begin{aligned}(\text{P0.b}) : doable(a) \Leftarrow\ & subject(u) \wedge own(u, att) \wedge action(manage(u, att), a) \\ & \wedge\ attributes(att, name) \wedge attributes(att, address).\end{aligned}$$

In the form above, *att* denotes personal attributes and $attributes(att, type)$ indicates that *att* includes data type *type*. Axiom $action(manage(u, att), a)$ denotes that action a is defined as a set of actions for managing *att* including "read" and "write" actions. Here, DS u has privileges to manage his or her own name and address.

In contrast, a digital content owner can state that no subscribers have any right to propagate the content in the following form, which has been used as an example:

$$\begin{aligned}(\text{P0.c}) : \neg doable(a) \Leftarrow\ & subject(u_1) \wedge subscriber(u_2) \wedge \neg subscriber(u_3) \\ & \wedge own(u_1, cont) \wedge action(send(u_2, u_3, cont), a),\end{aligned}$$

where $subscriber(u)$ denotes a subscriber of the digital content. $own(u_1, cont)$ indicates that DS u_1 owns digital content *cont*. $send(u_2, u_3, cont)$ corresponds to an action where u_2 propagates digital content *cont* to u_3.

Domain Policy Creation. Domain policies are created using meta-domain policies from the meta-domain or other domains. Algorithm 1 shows the procedure for a domain that creates domain policies whose meta domain is e_m.

An array, $pols^{(d)}$, is initially created by calling a utility function, `new_array`(), for storing a list of policies (step 1). e_m's meta-domain policies are stored in a list of policies $pols^{(m)}$ by calling `getMetaDomainPolicies`(e_m) that return the meta-policies obtained from a database in this domain or retrieve them from e_m (step 2). The appropriateness of all meta-domain policies in schema and content is examined (step 3). Function `verify`(p_i) examines whether policies are valid comparing them with meta-policies such as the ones from the root domain if they can be obtained (step 4). If a policy is verified, it is instantiated by calling function `instantiate`($\cdot$) as a new policy encoding abstract axioms in concrete data types and action types used in this domain (step 5), and it is registered in the policy list (step 6). For example, an instantiated policy applying the above approach to policy P0.a is

$$\begin{aligned}(\text{P1}) : \neg doable(a) \Leftarrow\ & subject(u_1) \wedge subject(u_2) \wedge own(u_1, data) \\ & \wedge action(send(u_1, u_2, data), a) \wedge \neg consent(u_1, a) \\ & \wedge data\_type(data, medical\_records).\end{aligned}$$

In policy P1, axiom $data\_type(\cdot)$ is added to constrain the specified data type of personal data. Here, the administrator of this domain restricts the propagation of personal medical records without obtaining the DS's consent strengthening the meta-domain policies as baseline policies.

Next, the administrator of this domain creates a list of domain specific policies *pols* by calling function `createDomainSpecificPolicies`($\cdot$) (step 9). Each policy created in the above step is verified (step 11) and its relationship with existing policies is checked. The relationships between two policies are listed in Fig. 3 [13].

Algorithm 1. `createDomainPolicies`(e_m)

```
1: pols^(d) ← new_array()
2: pols^(m) ← getMetaDomainPolicies(e_m)
3: for all i such that p_i ∈ pols^(m) do
4:    if verify(p_i) = true, then
5:       p_i ← instantiate(p_i)
6:       pols^(d).add(p_i)
7:    end if
8: end for
9: pols ← createDomainSpecificPolicies()
10: for all j such that p_j ∈ pols do
11:    if verify(p_j) = true, then
12:       for all k such that p_k ∈ pols^(d) do
13:          if p_k.diverges(p_j), then
14:             pols^(d).del(p_k)
15:          else if p_k.extends(p_j), then
16:             pols^(d).del(p_k)
17:             pols^(d).add(p_j)
18:          else if p_k.shuffles(p_j), then
19:             p ← createMeetPolicy(p_k, p_j)
20:             pols^(d).del(p_k)
21:             pols^(d).add(p)
22:          end if
23:       end for
24:    end if
25: end for
26: return pols^(d)
```

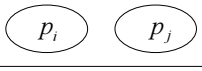
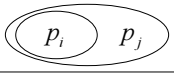
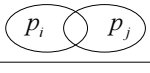
Policy Similarity Type	Authorized Requests
p_i Converges p_j	
p_i Diverges p_j	
p_i Restricts p_j	
p_i Extends p_j	
p_i Shuffles p_j	

Fig. 3. Similarity Types of Policies

There are five types of similarity in policies, "converges", "diverges", "restricts", "extends", and "shuffles". Note that the similarity between two policies is viewed with respect to which of their conditions hold and that the area covered by a circle representing a policy corresponds to the scope with which it grants execution. Of these, the relationship between an existing policy instantiated from a meta-policy and a new policy created as being domain specific one corresponds to "diverges" (step 13); the existing policy is registered (step 14), since the policy has a different constraint from that of the existing one. If the relationship corresponds to "extends" (step 15), the existing policy is deleted and the new policy is registered (steps 16–17). Otherwise, if the two policies are in a "shuffles" relationship (step 18), a policy for the union of the existing and the new one is newly created by calling function `createMeetPolicy`$(\cdot)$ (step 19); the existing one is deleted (step 20), and the new intersection policy is added into the list of domain policies (step 21). Finally, this function returns a set of registered domain policies (step 26). Note that the above approach to integrating policies strengthens policy constraints except for "converges" and "restricts" cases in which the existing policy encompasses the new one.

Subject Policy Creation. Subject policies in a domain are policies for a particular DS restricting a specific type of the DS's personal data within a certain context.

If the data are created by the domain by which they are managed or if the DS owns the data and delegates their management to the domain, the subject policies are created using the domain policies. The domain policies are instantiated for the DS in the same way as described by Algorithm 1. For example, policy P2 has an application for the same approach and modus ponens to policy P1.

$$\begin{aligned}(\mathrm{P2}) : \neg doable(a) \Leftarrow\ & subject(Alice) \wedge subject(Bob) \wedge own(Alice, data) \\ & \wedge\ action(send(Alice, Bob, data), a) \wedge \neg consent(Alice, a) \\ & \wedge\ data\_type(data, medical\_records).\end{aligned}$$

In contrast, if the data originally propagate from a different domain acting as a DC to a domain acting as a DP, the agreed upon policies are provided by the DC as a result of the policy being adjusted between the two domains. Here, the DP domain incorporates the agreed upon policies into their subject policies to handle the data. The detailed on the procedure are explained in the following section.

4.2 Policy Agreement between Administrative Domains

Algorithm 2 has the procedure for DP entity e_p requesting and obtaining DS u's personal data whose attribute type is at from DC entity e_c.

DP e_p initially retrieves the subject policies for DC e_c (controller policies) by calling `getControllorPolicies`$(\cdot)$ (step 1). e_p obtains its subject policies (processor policies) for data type at and DS u by calling `getProcessorPolicies`$(\cdot)$

Algorithm 2. `getDataWithPolicies`(at, u, e_p, e_c)

1: $pols^{(c)} \leftarrow$ `getControllorPolicies`(at, u, e_p, e_c)
2: $pols^{(p)} \leftarrow$ `getProcessorPolicies`(at, u)
3: **for all** j such that $p_j \in pols^{(p)}$ **do**
4: **if** $\neg(\exists p \in pols^{(c)})$; $p.converges(p_j)$ or $p.extends(p_j)$, **then**
5: **return** *null*
6: **end if**
7: **end for**
8: $pols^{(a)} \leftarrow$ `getAgreedPolicies`$(at, u, pols^{(p)}, e_p, e_c)$
9: **for all** k such that $p_k \in pols^{(a)}$ **do**
10: **if** `verify`(p_k) = false or $\neg(\exists q \in pols^{(p)})$; $q.converges(p_k)$, **then**
11: **return** *null*
12: **end if**
13: **end for**
14: `save`$(pols^{(a)})$
15: $hash\_val \leftarrow \mathtt{HMAC}_{\mathtt{K}}(pols^{(a)}.id \mid\mid e_p)$
16: $data \leftarrow$ `getData`$(at, u, e_p, e_c, hash\_val)$
17: **return** *data*

(step 2). Every policy contained in $pols^{(p)}$ is checked whether it has a "converges" or "extends" relationship with a policy contained in $pols^{(c)}$ or not, since e_p needs to comply with the controller policies that e_c presents (steps 3–7). If no processor policy satisfies a controller policy, this function returns null and e_p cannot obtain the requested data since e_p and e_c cannot agree on how data are handled.

If the processor policies are appropriate, e_p sends a request to e_c for policy agreement on handling the specified data invoking `getAgreedPolicies`$(\cdot)$ and obtains agreed upon policies $pols^{(a)}$ (step 8). e_p verifies the agreed upon policies and checks whether all policies contained in $pols^{(a)}$ are the same as policies $pols^{(p)}$ or not (steps 9–13). If the agreed upon policies are appropriate, e_p stores them to confirm the contract on the personal information practices between e_p and e_c (step 14).

The procedure for data retrieval is executed in steps 15–16. A keyed hashing for message authentication code (HMAC) [16] mechanism is used to access the data. $\mathtt{HMAC}_{\mathtt{K}}(\cdot)$ indicates a HMAC function returning a hash value. Operator "||" stands for a concatenation of two strings. Here, the strings of the identifier of agreed upon policies $pols^{(a)}$ and e_p are concatenated and given as input to the above function (step 15). This hash value, as well as at, u, e_p, and e_c, is required as a credential to call function `getData`$(\cdot)$, in which e_p sends a request to e_c for u's data at and obtains the specified data from e_c (steps 16–17).

This credential-issuing scheme based on agreement enables fine-grained control of access, since the credential required for data cannot be obtained until its requestor has agreed on data practices.

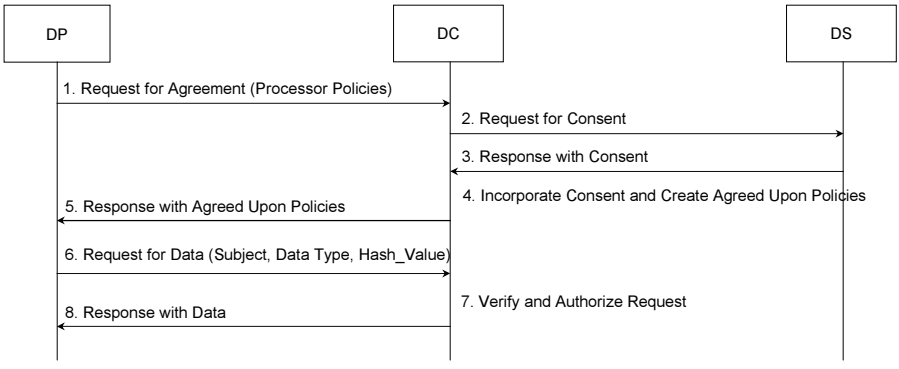


Fig. 4. Data Controller's Policy and Procedure for Data Provisioning

4.3 Control of Data Access and Incorporation of Subject Preferences

A domain's subject policies are generally not sufficient to enforce them when the domain attempts to make a decision to authorize a data access request because it is difficult for a domain's administrators to statically specify any constraints dependent on data types and DSs, or because it is impossible for a DS to specify his or her subject preferences unless conditions are presented. If the policies include a DS's decision or consent, the domain especially needs to interact with the DS and obtain it at runtime. Since this DS's decision or consent indicates one of the DS's subject preferences, they need to be incorporated into a domain's subject policies.

Figure 4 outlines the procedure when a DC receives requests from a DP in accordance with Algorithm 2. The procedure includes a DC's control to access managed personal data and dynamically incorporate the subject preferences of the DS.

When the DP sends a request for attaching an agreement to its processor policies (step 1), the DC verifies the request message and checks with its controller policies, and requests for a consent to the DS that owns the target data to consent by describing how the data will be used by the DP (step 2). The DS shows the consent to the DC, which is one of the subject preferences. Therefore, the consent is incorporated into controller policies and agreed upon policies are created based on the updated controller policies (step 4). The policies are agreements including an identifier that can be used to identify the authenticity of the data request arriving from the DP. After the DC responds with the above agreed upon policies (step 5), the DP makes a new request for the DS's personal data with a hash value (step 6). This value is verified to authenticate the received request from the DP, reproducing a value with the received identifiers. The use of HMAC protects the resource from snooping and extension attacks (step 7). Finally, the requested data are propagated to the DP (step 8).

In the above example, a DC incorporates a DS's consent into its subject policies at runtime when a decision on access control to propagate data is needed. In the same way, a DP also incorporates a DS's consent to process the DS's data at runtime when the DP attempts to do so.

4.4 Protection from Unauthorized Policy Updates

DC needs to have an access control mechanism for unauthorized policy updates and modifications. Here, the subject policies of DCs are regarded as restricted resources in the same way as personal data at DC.

Whenever a DC deploys new policies such as agreed upon policies and domain policies, it verifies their validity and authenticity. If the DC finds any inappropriateness in policies, it stops integrating them to mitigate the risk of unfair information practices.

5 Discussion

This section discusses several topics and issues related to the proposed model and framework.

5.1 Retaining Data Management

Policies in the proposed model that reflect local laws or regulations of social organizations or computer systems are appropriately provided beyond administrative domains in distributed environments since the model has a hierarchical structure for creating and propagating policies and supports a policy propagation chain in accordance with data propagation. By means of this approach, the administrator of a root domain can retain the management of data enforcing their provisioned policies that reflect his or her intentions regarding data governance even after the data have been propagated in the distributed system. In addition, participating parties in the system can clarify their liabilities concerning data practices and improve the accountability of their activities on handling data. To further support the lifecycle management of data and policies, a mechanism for updating provisioned policies at runtime is needed if it is difficult for an administrator to statically specify policies containing various types of constraints.

5.2 Policy Conflicts

The proposed model facilitates the avoidance of policy conflicts between entities detecting similarities between policies when new policies are created or integrated. Since DP entities need to accept DC's controlling policies or strengthen their constraints, there is no room for policy negotiation in policy between them. This is appropriate from the governance and compliance viewpoints of a root domain's administrator. However, this approach may reduce the expressiveness of policies when flexible representations such as exceptional action rules are needed.

Although this is a problem involving a trade-off, finding an appropriate balance between administrative restrictive descriptions and rich representations of policies is an open issue that needs further investigation.

5.3 Directive and Recommendation Policies

As explained in Section 4.1, governance policies state that personal information can be managed by DS as its owner from a user-centric perspective. However, it is difficult for a DS to take an appropriate action in all environments. For example, privacy laws and privacy guidelines such as those of the OECD [17] dictate that enterprises should take into account the consent given by people to use their data for specified purposes. However, people may possibly act inappropriately if they do not understand what their consent involves, which unfortunately does not match the intentions of legislators of privacy-related guidelines. In such cases, administrators can specify complementary directive policies stating that enterprises should provide sufficient explanations to people about how their data are used or that enterprises should suggest to them possible actions that can be taken.

6 Conclusion and Future Work

This paper described a policy provisioning model in which distinct entities in distributed environments collaboratively create and propagate data handling policies. Algorithms for creating and integrating policies enable data handling policies to be deployed and enforced to securely and privacy control access to personal data in an enhancing manner. The proposed framework helps to manage the lifecycle of personal data using their handling policies that reflect the intentions of the system administrator and their owner. Future work includes that on updating policies and resolving conflicts in them.

References

1. Byun, J.W., Bertino, E., Li, N.: Purpose Based Access Control of Complex Data for Privacy Protection. In: Proceedings of the 10th ACM Symposium on Access Control Models and Technologies (SACMAT 2005), pp. 102–110 (2005)
2. Ardagna, C.A., Cremonini, M., De Capitani di Vimercati, S., Samarati, P.: A Privacy-Aware Access Control System. Journal of Computer Security 16(4), 369–397 (2008)
3. Gomi, H.: A Persistent Data Tracking Mechanism for User-Centric Identity Governance. Identity in the Information Society (March 2010), doi:10.1007/s12394-010-0069-4
4. Schneck, P.: Persistent Access Control to Prevent Piracy of Digital Information. Proceedings of the IEEE 87(7), 1239–1250 (1999)
5. Sibert, O., Bernstein, D., Wie, D.: DigiBox: A Self-Protecting Container for Information Commerce. In: Proceedings of the 1st Conference on USENIX Workshop on Electronic Commerce (WOEC 1995), p. 15 (1995)

6. Karjoth, G., Schunter, M., Waidner, M.: Platform for Enterprise Privacy Practices: Privacy-Enabled Management of Customer Data. In: Dingledine, R., Syverson, P.F. (eds.) PET 2002. LNCS, vol. 2482, pp. 69–84. Springer, Heidelberg (2003)
7. Casassa Mont, M., Pearson, S., Bramhall, P.: Towards Accountable Management of Identity Privacy: Sticky Policies and Enforceable Tracing Services. In: Mařík, V., Štěpánková, O., Retschitzegger, W. (eds.) DEXA 2003. LNCS, vol. 2736, pp. 377–382. Springer, Heidelberg (2003)
8. Liberty Alliance Project: Liberty IGF Privacy Constraints Specification (2008), http://www.projectliberty.org/specs
9. W3C: The Platform for Privacy Preferences 1.0 (P3P1.0) Specification (2002), http://www.w3.org/TR/P3P/
10. W3C: A P3P Preference Exchange Language 1.0 (APPEL1.0) (2002), http://www.w3.org/TR/P3P-preferences/
11. OASIS: eXtensible Access Control Markup Language, XACML (2005)
12. IBM: Enterprise Privacy Authorization Language (EPAL 1.2) (2003), http://www.w3.org/Submission/2003/SUBM-EPAL-20031110/
13. Mazzoleni, P., Crispo, B., Sivasubramanian, S., Bertino, E.: XACML Policy Integration Algorithms. ACM Transactions on Information and System Securiry 11(1), 1–29 (2008)
14. Belokosztolszki, A., Moody, K.: Meta-Policies for Distributed Role-Based Access Control Systems. In: Proceedings of the Third International Workshop on Policies for Distributed Systems and Networks (POLICY 2002), pp. 3–18 (2002)
15. Bettini, C., Jajodia, S., Sean Wang, X., Wijesekera, D.: Provisions and Obligations in Policy Management and Security Applications. In: Proceedings of the 28th International Conference on Very Large Data Bases (VLDB 2002), pp. 502–513 (2002)
16. Krawczyk, H., Bellare, M., Canetti, R.: HMAC: Keyed-Hashing for Message Authentication, RFC 2104 (1997)
17. OECD: OECD Guidelines on the Protection of Privacy and Transborder Flows of Personal Data (2004), http://www.oecd.org/document/18/0,2340,en_2649_201185_1815186_1_1_1_1,00.html