Policy-based Authentication and Authorization based on the Layered Privacy Language

Sebastian Wilhelm¹, Armin Gerl²

Abstract: In 2018 the General Data Protection Regulation (GDPR) has been enforced providing a new legal framework with rules and regulations for processing personal data. The requirement for distinguishing between purposes has been introduced, leading to the necessity of adapting existing authentication and authorization processes. We introduce a detailed authentication and authorization extension, which is able to verify requests on personal data based on the Layered Privacy Language (LPL). This extension is evaluated in the form of a benchmark, utilizing the Policy-based De-identification, to demonstrating its efficiency and suitability for data-warehouses.

Keywords: Access Control, GDPR, Privacy, Privacy Language

1 Introduction

The General Data Protection Regulation (GDPR) has been enforced in Europe since May 25th 2018 constituting Privacy by Design and Privacy by Default for all technical systems [Co16, Art. 25]. Hereby, processing of personal data, which also includes storage of personal data in databases and data-warehouses, requires either a legal basis or consent [Co16, Art. 6]. Furthermore, several conditions for consent have to be fulfilled, including the necessity of the Controller demonstrating that the Data Subject consented. Consent has to be given freely, or consent has to be differentiated according to the purposes of processing [Co16, Art. 7, Recital 32]. Therefore, common authentication and authorization processes have to be adapted and extended to allow a purpose-based processing of personal data to comply with the GDPR. The Layered Privacy Language (LPL) in combination with its overarching privacy framework intends to comply with the requirements of the GDPR to enforce privacy policies 'from consent to processing' [Ge18b].

The focus of this work lies in the detailing of the extension of authentication and authorization by introducing purpose- and data-based authorization based on LPL as well as its evaluation. The evaluation considers the scalability of this extension utilizing a benchmark.

¹ Deggendorf Institute of Technology, Technology Campus Grafenau, Hauptstraße 3, D-94481 Grafenau sebastian.wilhelm@th-deg.de
² University of Passau, Chair of Distributed Information Systems, Innstraße 41, D-94032 Passau, armin.gerl@uni-passau.de

doi:10.18420/btw2019-ws-25
In the following, we will give an introduction of relevant elements of LPL in section 2. Section 3 details the authentication and authorization processes of LPL introducing purpose- and data-based authorization, which will be evaluated in section 4. Related work is presented in section 5. Lastly, the work is concluded and future work is outlined in section 6.

2 Layered Privacy Language (LPL)

The Layered Privacy Language (LPL) is a domain specific language, designed to model legal privacy policies [Ge18b]. In the following, we will detail only the relevant elements for the scope of this paper. We also want to point out, that we will not consider the UI Extension [Ge18a] or Art. 12-14 GDPR Extension [GP18].

The root element of LPL is the LayeredPrivacyPolicy-element $lpp$, which has a set of Purpose-elements $p$, a DataSource-element $ds$, as well as several attributes which are not relevant for the scope of this work. Hereby, the DataSource-element denotes either the Data Subject or another legal entity (e.g. a Controller) providing the data. A Purpose-element further describes a set of DataRecipient-elements $dr$ as well as a set of Data-elements $d$, further elements and attributes are omitted for the sake of this work.

Based on LPL an overarching privacy framework is developed, which is intended to provide both a user interface to inform the user on the privacy policy as well as the enforcement of the policy. In general, the life-cycle of LPL consists of six steps. It starts with the Creation step, in which the responsible data protection officer creates the LPL privacy policy. This policy is then presented to the Data Subject during the Negotiation step, in which the Data Subject can personalize it and eventually decides to accept or give consent to the personalized privacy policy or not. If the privacy policy is accepted or consented to, then the privacy policy will be validated and information about the Data Subject and its personal data will be added during the Pre-Processing before it is stored along with the regular data during Storage. During the Transfer step, data may be transferred to third parties under a negotiated policy, to which then the original privacy policy will be added. Lastly, the stored data will be processed during the Usage step, which will be the main focus in this work.

Hereby, the Policy-based De-identification process will be applied on each request for data processing, whereas data will be de-identified as necessary, when the requesting entity is authenticated and authorized to process the data for the specified purposes. The processes for authentication and authorization are the focus of this work and will be detailed in the following in the context of the Policy-based De-identification.

3 Policy-based De-Identification

A request on the Policy-based De-identification generally takes the form of the following tuple:

$$request = (user.Identifier, credential, \hat{P}, \hat{D}, \hat{DS})$$
Where \texttt{userIdentifier} is the unique identifier of the requesting user (e.g. username), \texttt{credential} the credential of the user for authentication (e.g. password), $\hat{P}$ the set of all requested 
\texttt{Purposes}, $\hat{D}$ the set of all requested \texttt{Data} and $\hat{DS}$ the set of all requested \texttt{DataSources}.

LPL now defines four steps for this process of inspecting which \texttt{Data} the \texttt{Data Recipient} is allowed to request:

1. **Entity-Authentication.** Verify the authenticity of the requesting entity.

2. **Purpose-Authorization.** Get all \texttt{Child-Purposes} $\hat{P}_{\text{child}}$ of the requested \texttt{Purposes} $\hat{P}_{\text{requested}}$ and consider which \texttt{Purposes} are relevant.

3. **Entity-Authorization.** Verify if the requesting entity (\texttt{Data Recipient}) is authorized to request the \texttt{Purposes} $\hat{P}_{\text{requested}}$ (or $\hat{P}_{\text{child}}$).

4. **Data-Authorization.** Verify whether it is allowed to request the requested \texttt{Data} $\hat{D}$ of the requested \texttt{Data Sources} $\hat{DS}$ with the authorized \texttt{Purposes} $\hat{P}_{\text{authorized}}$.

The result of the de-identification process can be described as follows:

\[
\text{result set} = (\hat{SD}); \quad \hat{SD} = (\text{dataSource}, \hat{DP}); \quad \hat{DP} = (\text{data}, \hat{P}_{\text{result}})
\]

Where \texttt{dataSource} is a \texttt{DataSource}-element, \texttt{data} a requested and authorized \texttt{Data}-element, and $\hat{P}_{\text{result}}$ is the set of all relevant \texttt{Purposes} for a specific \texttt{Data}-element for a specific \texttt{Data Source}-element which are authorized. $\hat{DP}$ maps all relevant \texttt{Purposes} to a \texttt{Data}-element and $\hat{SD}$ maps this \texttt{Data-Purpose Maps} to a specific \texttt{Data Source}-element. The four steps of the Policy-based De-identification will be detailed in the following.

### 3.1 Entity-Authentication

The \texttt{Entity-Authentication} process has the task of verifying the identity of a \texttt{Data Recipient}. It will be inspected if the requesting entity is known\(^3\). Then the entity authentication will be conducted, based on the configured authentication method. A request on the \texttt{Entity-Authentication} has generally the form of the following tuple:

\[
\text{request} = (\text{userIdentifier}, \text{credential})
\]

The module verifies if the handed \texttt{userIdentifier} and the \texttt{credential} belongs together. Some different authentication methods are possible (e.g. hashed passwords or OAuth). The result of \texttt{Entity-Authentication} is the authenticated \texttt{DataRecipient}-element $dr_{\text{auth}}$.

\(^3\) Basically, an entity is stored in the data-storage (e.g. database). But there can be some special cases when entities are known but not stored in the data-storage (e.g. for some API-keys)
3.2 Purpose-Authorization

The Purpose-Authorization process has the task of gathering all authorized Purposes for a specific request. Each Purpose $p$ can have Child-Purposes $\hat{P}_{\text{child}}$. Contrary to the original definition in [Ge18b], it will be allowed that a Child-Purpose inherits from two (or more) Parent-Purposes. This change in definition should provide the framework with more flexibility. It is important, that no cycles in the definition of the Purposes exists.

A Purpose $p$ is relevant for a DataSource-element $ds$ if it (or one of its Parent-Purposes) is part of the privacy policy of the Data Source. If a Purpose is mentioned in at least one of the requested DataSource-elements $DS$, it is relevant for the current request. A request on Purpose-Authorization has generally the form of the following tuple:

$$\text{request} = (\hat{P}_{\text{requested}}, \hat{DS}_{\text{requested}})$$

Where $\hat{P}_{\text{requested}}$ is the set of all requested Purpose-elements and $\hat{DS}_{\text{requested}}$ is the set of all requested DataSource-elements. The module initially receives all inherited child Purpose-elements $P_{\text{child}}$ of the requested Purpose-elements $\hat{P}_{\text{requested}}$ and then maps to each of the inherited child Purpose-elements $P_{\text{child}}$ all DataSource-elements for which this specific Purpose-element is relevant. Finally, all elements from $\hat{P}_{\text{child}}$, which do not have mapped any DataSource-elements, will be removed. The result of the Purpose-Authorization can be described as follows:

$$\text{resultset} = (\hat{PD}); \quad \hat{PD} = (p, \hat{E})$$

Where $p$ is the Purpose-element and $\hat{E}$ is the set of DataSource-elements for which the corresponding Purpose-element is relevant.

3.3 Entity-Authorization

The Entity-Authorization process verifies whether a DataRecipient $dr$ is authorized for requesting a set of Purposes $\hat{P}$. Each Entity $e$ can have Child-Entities $\hat{E}_{\text{child}}$. So, the Entity $e$ can inherit the rights for requesting Purposes from its Child-Entities $\hat{E}_{\text{child}}$. It is important, that no cycles in the definition of the Entities exists.

A DataRecipient $dr$ is allowed to request a specific Purpose $p$ if the DataRecipient $dr$ or at least one of its children $\hat{DR}_{\text{child}}$ is authorized for using the Purpose $p$. A request on Entity-Authorization has generally the form of the following tuple:

$$\text{request} = (dr, \hat{P}_{\text{relevant}}, \hat{P}_{\text{requested}})$$
Where \( dr \) is the DataRecipient, \( P_{\text{relevant}} \) is the set of all relevant Purposes and \( P_{\text{requested}} \) is the set of all requested Purposes. The result of Entity-Authorization is a set of all relevant Purposes for which the Data Recipient is authorized.

### 3.4 Data-Authorization

Data-Authorization verifies whether it is allowed to request the Data \( \hat{D} \) of the DataSource-elements \( \hat{DS} \) with the authorized Purposes \( \hat{P}_{\text{aut}} \). For each Purpose \( p \) it is defined which Data elements \( \hat{D} \) are allowed to be requested for the Purpose \( p \) and for each DataSource Purposes are defined to be used. The result of Purpose-Authorization has already mapped each requested DataSource-element \( \hat{DS} \) to its relevant Purposes. The Data-Authorization now has to reorder the map and get for each requested DataSource \( ds \) and each requested Data element \( d \) a set of possible Purposes. A request on Data-Authorization has generally the form of the following tuple:

\[
\text{request} = (\hat{D}, \hat{DS}, \hat{P}_{\text{aut}})
\]

Where \( \hat{D} \) is the set of all requested Data, \( \hat{DS} \) is the set of all requested DataSource and \( \hat{P}_{\text{aut}} \) ist the set of all authorized Purposes. Basically, the module reorder the data \( \hat{D}, \hat{DS} \) and \( \hat{P}_{\text{aut}} \) so that they are in the form of the resultset. Some different configurations of Data-Authorization are possible (e.g. return only data which are permitted for all requested DataSource elements). The result of the Data-Authorization can be described as follows:

\[
\text{resultset} = (\hat{SDP}); \quad \text{SDP} = (ds, \hat{DP}); \quad \text{DP} = (d, \hat{P})
\]

Where \( p \) is the Purpose-element, \( ds \) the DataSource-element, \( d \) the Data-element and \( \hat{E} \) the set of entities for which the corresponding Purpose-element is relevant.

### 4 Benchmark-Evaluation

We implemented the steps Entity-Authentication, Purpose-Authorization, Entity-Authorization and Data-Authorization on the basis of section 3 in JAVA and verified the functionality with unit tests, to measure the execution times of the individual steps depending on the input variables (e.g. number of purposes) to determine the efficiency of the Policy-based De-identification. It is not the goal to be able to calculate the detailed execution times of specific requests on a dataset with specific parameters rather it is important for using the Layered Privacy Language on data-warehouses to ensure that the concept is scaleable. In this section, execution times are analyzed, time intensive processes and method are to be found and if necessary and possible, improved or possible improvements found.

\[ P_{\text{requested}} \] will be needed for a specific system configuration where will be considered if the permission on at least one of the requested Purposes \( P_{\text{requested}} \) is missing.
The execution time of a request basically depends on the system-configuration (e.g. type of entity authentication), the amount and dependencies of the data in the data-storage (e.g. number of possible Data Subjects) and the amount of the requested Data Sources, Data and Purposes. The goal is now to measure and compare the execution times of the individual steps while varying the individual parameters. For this purpose, it is necessary to implement a Benchmark Test Suite which tests all possible system configurations by entering the parameters for the data store and the requested data and then saves the individual execution times. For this the Mockito framework is used, to mock the interface to the persistence layer. Specifically, we determine the parameters in Tab. 1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PurposeAmount</td>
<td>Amount of different Purposes, which are predetermined.</td>
</tr>
<tr>
<td>PurposeBranching</td>
<td>Degree of branching of the Purposes with each other.</td>
</tr>
<tr>
<td>DataSourceNumber</td>
<td>Amount of different entities to which Data are stored.</td>
</tr>
<tr>
<td>PurposePerLPP</td>
<td>Amount of Purposes for which a single entity has consented to.</td>
</tr>
<tr>
<td>DataNumber</td>
<td>Value determining the amount of Data elements.</td>
</tr>
<tr>
<td>DataPerPurpose</td>
<td>Value determining the amount of Data elements which can be requested with a single, specific Purpose.</td>
</tr>
<tr>
<td>DataRecipientAmount</td>
<td>Amount of entities which are authorized to request data.</td>
</tr>
<tr>
<td>RecipientBranching</td>
<td>Degree of branching of the Data Recipients.</td>
</tr>
<tr>
<td>PurposePerRecipient</td>
<td>Amount of Purposes for which a single Data Recipient is authorized.</td>
</tr>
<tr>
<td>RequestedPurposes</td>
<td>Amount of requested Purposes.</td>
</tr>
<tr>
<td>RequestedData</td>
<td>Amount of different requested Data elements.</td>
</tr>
<tr>
<td>RequestedDataSource</td>
<td>Amount of requested Data Sources.</td>
</tr>
</tbody>
</table>

Tab. 1: Configuration parameters for benchmark evaluation.

The Benchmark Test Suite first mocks the complete data store of the system depending on the configuration parameters. From this mocked objects, data (Purposes, Data, Data Sources) are determined, again depending on the configuration parameters, which should be used for a request to the system. This request then will be executed for every possible system configuration, measuring how much time each step will take. Finally, all measurements are stored in a separate, external database. All tests were executed on a MacBook Pro (13 inch, Mid 2012) with the operating system macOS High Sierra (10.13.3), a 2.5 GHz Intel Core i5 processor and 16 GB 1600 MHz DDR3 random access memory.

The measurements were analyzed by comparing the change of the execution times by varying one single configuration parameter at a time. Tab. 2 shows an example of such values. With this (and further) measurements it was possible to approximate algorithms to determine a dependency on the varied configuration parameter.

It can be observed that Entity-Authentication is affected by DataRecipientAmount, Purpose-Authentication is affected by PurposeAmount, DataSourceNumber, PurposePerLPP, RequestedPurposes and RequestedDataSource, Entity-Authentication is affected by PurposeAmount, DataSourceNumber, PurposePerLPP and DataRecipientAmount, Data-Authentication is af-

---

5 The persistence-layer of the system, holing the relevant user and meta data for the Layered Privacy Language.
Policy-based Authentication and Authorization based on LPL 7

<table>
<thead>
<tr>
<th>RequestedDataSource</th>
<th>1</th>
<th>100</th>
<th>250</th>
<th>500</th>
<th>1000</th>
</tr>
</thead>
<tbody>
<tr>
<td>Entity Authentication</td>
<td>0.38</td>
<td>0.38</td>
<td>0.5</td>
<td>0.41</td>
<td>0.37</td>
</tr>
<tr>
<td>Purpose Authorization</td>
<td>110.76</td>
<td>251.21</td>
<td>494.89</td>
<td>878.67</td>
<td>1675.47</td>
</tr>
<tr>
<td>Entity Authorization</td>
<td>1.59</td>
<td>1.84</td>
<td>1.82</td>
<td>1.89</td>
<td>1.88</td>
</tr>
<tr>
<td>Data Authorization</td>
<td>0.77</td>
<td>0.72</td>
<td>0.85</td>
<td>0.73</td>
<td>0.68</td>
</tr>
<tr>
<td>Total</td>
<td>113.6</td>
<td>254.25</td>
<td>498.16</td>
<td>881.84</td>
<td>1678.51</td>
</tr>
</tbody>
</table>

Tab. 2: Measured execution time in ms of the individual steps on varying the RequestedDataSource.

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>PurposeAmount</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PurposeBranching</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataSourceNumber</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PurposePerLpp</td>
<td>✓</td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DataNumber</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>DataPerPurpose</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>DataRecipientAmount</td>
<td>✓</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>RecipientBranching</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>RequestedPurposes</td>
<td></td>
<td>✓</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RequestedData</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
<tr>
<td>RequestedDataSource</td>
<td></td>
<td></td>
<td></td>
<td>✓</td>
</tr>
</tbody>
</table>

Tab. 3: Relevance of the individual benchmark test parameters to the different steps.
fected by DataNumber, DataPerPurpose, RequestedPurposes and RequestedData (see Tab. 3). In summary, it could be observed that each of the benchmark parameters has at a maximum a linear effect on the execution time of the steps. This ensures scalability.

However, the parameters DataSourceNumber, RequestedPurposes and RequestedDataSources are noteworthy since they already have an execution time $>$100ms in some tested scenarios. Which is unacceptable for a system that is intended to work in real-time, therefore further investigation research was executed.

In the following, the parameter RequestedDataSources is exemplary further analyzed. The measured values of the benchmark evaluation indicate the RequestedDataSource parameter has a direct, linear effect on the execution time of Purpose Authorization.

$$t_{\text{purposeAuthorization}} = 1.5708 ms \times \text{RequestedDataSources} + 100.69 ms$$

The value RequestedDataSources determines the number of DataSource-elements from which data should be queried. A request of 10,000 elements and even bigger numbers are normal in data-warehouse scenario (e.g. for generating a statistic). With the given formula, we already get the execution times for a RequestedDataSource of 10,000.

$$t_{\text{purposeAuthorization}} = 15808.69 ms$$

These execution times for the step of PurposeAuthorization is not acceptable for a real-time system. In order to optimize the software design, additional measurements were taken and analyzed.

<table>
<thead>
<tr>
<th>Step Description</th>
<th>Execution Time</th>
</tr>
</thead>
<tbody>
<tr>
<td>Put all requested and inherited Purposes to the key-set</td>
<td>7 ms</td>
</tr>
<tr>
<td>of which executing getInheritedPurposes</td>
<td>7 ms</td>
</tr>
<tr>
<td>Put the Data Sources to the relevant Purposes</td>
<td>1289 ms</td>
</tr>
<tr>
<td>of which executing getLppRelevantPurposes</td>
<td>1278 ms</td>
</tr>
<tr>
<td>Getting all not relevant Purposes of the key-set</td>
<td>&lt;1 ms</td>
</tr>
<tr>
<td>Remove all not relevant Purposes from the result map</td>
<td>&lt;1 ms</td>
</tr>
<tr>
<td><strong>Total execution time</strong></td>
<td><strong>1296 ms</strong></td>
</tr>
</tbody>
</table>

Tab. 4: Detailed measured execution times of Purpose-Authorization for the benchmark test parameter RequestedDataSource.

From the 1296 ms execution time of the step Purpose-Authorization, are 1286 ms waiting for methods of other modules (PurposeRequestManager - the mocked persistence layer). The pure computing time required by this step is approximately 10 ms.

Thus, we note that the high measured execution times of the RequestedDataSource parameter are due only to the time Mockito takes to process the getInheritedPurposes and getLppRelevantPurposes methods. Further optimization of Purpose-Authorization does not seem necessary to solve the original problem of long execution times for a high amount of RequestedDataSource. By connecting the implementation to a high-performance persistence layer, the execution time of the steps promise to be within an acceptable range. This
determination can analogously be transferred to the parameters $\text{DataSourceNumber}$ and $\text{RequestedPurposes}$.

In summary, it can thus be stated that the measured execution times are very much influenced by $\text{Mockito}$, and that queries can be processed even faster by a high-performance persistence layer. The aim of this section 4, however, was merely to show that the implementation can also be used in large systems and is scalable. This could be demonstrated.

5 Related Work

Data protection and purpose-based storage and processing of personal data is not a new field of research. Thereby, some privacy languages were already proposed, each with their own distinct application [Ge18b]. In the context of authentication and authorization amongst other the privacy languages AIR [Kh10], DORIS [BB88], E-P3P [As02], P2U [IV14], P3P [Cr06], Ponder [SLL01], Primelife policy language [Ar09], Rei [Ka02], SecPAL [BFG10], SPECIAL [Ki18], XACL [BCF04], are worth mentioning.

But also in the context of database systems, purpose-based authorization has been introduced in Hippocratic Databases [Ag02], which we will detail in the following. In contrast to the Layered Privacy Language, the Hippocratic Database design is not to be considered as fixed technology or framework [vTJ11], but as a vision of a database system that implements the principles of Purpose Specification, Consent, Limited Collection, Limited Use, Limited Disclosure, Limited Retention, Accuracy, Safety, Openness and Compliance. In the concept of Hippocratic Databases, analogues to the Layered Privacy Language, Purposes are the central component. For the stored data, the Purposes is attached, which is relevant for this record, for example we look at the schema of Tab. 5. In contrast to the Layered Privacy Language, the Entity-Authentication is not part of the Hippocratic Database concept. Furthermore, the Layered Privacy Language promises a more flexible Data-Authorization with different configuration parameters (e.g. return only consented data from Data Subjects) and so a more flexible usage of the framework.

6 Conclusion

The Layered Privacy Language defines six steps to privacy-preserving process data - Entity-Authentication, Purpose-Authorization, Entity-Authorization, Data-Authorization, Minimum Anonymization and Application of Privacy Model. The first four steps are detailed and evaluated in this work. The benchmark evaluation shows that the implemented modules
are scalable and capable of processing complex requests (many requested data, extensive
data storage) efficient, which depends on an efficient backend.

For future works the remaining steps of the Policy-based De-identification process will be evaluated, showing the introduced overhead of the policy processing against the de-identification solely. Furthermore, we plan to introduce pseudonymization capabilities utilizing tokenization methodology in both LPL as well as the Policy-based De-identification process. Another concern is the possibility of privacy inference that has to be detected and prevented due to consecutive requests. Lastly, different scenarios concerning relevant privacy use cases, will be defined and used to evaluate the complete Policy-based De-identification process based on LPL.

References


[Ar09] Ardagna, Claudio A; Bussard, Laurent; De Capitani di Vimercati, Sabrina; Neven, Gregory; Pedrini, E; Paraboschi, S; Preiss, F; Samarati, P; Trabelsi, S; Verdicchio, M: Primelife policy language. In: W3C Workshop on Access Control Application Scenarios. W3C, 2009.


[Cr06] Cranor, Lorrie; Dobbs, Brooks; Egelman, Serge; Hogben, Giles; Humphrey, Jack; Langheinrich, Marc; Marchiori, Massimo; Presler-Marshall, Martin; Reagle, Joseph M.; Schunter, Matthias; Stampley, David A.; Wenning, Rigo : The Platform for Privacy Preferences 1.1 (P3P1.1) Specification. World Wide Web Consortium, Note NOTE-P3P11-20061113, November 2006.

Policy-based Authentication and Authorization based on LPL 11


