IEICE TRANS. INF. & SYST., VOL.E106–D, NO.9 SEPTEMBER 2023

PAPER Special Section on Information and Communication System Security

# Policy-Based Method for Applying OAuth 2.0-Based Security Profiles

**Takashi NORIMATSU**[†,††a)], **Yuichi NAKAMURA**[†], ***Nonmembers***, ***and*** **Toshihiro YAMAUCHI**[††], ***Member***

**SUMMARY** Two problems occur when an authorization server is utilized for a use case where a different security profile needs to be applied to a unique client request for accessing a distinct type of an API, such as open banking. A security profile can be applied to a client request by using the settings of an authorization server and client. However, this method can only apply the same security profile to all client requests. Therefore, multiple authorization servers or isolated environments, such as realms of an authorization server, are needed to apply a different security profile. However, this increases managerial costs for the authorization server administration. Moreover, new settings and logic need to be added to an authorization server if the existing client settings are inadequate for applying a security profile, which requires modification of an authorization server's source code. We aims to propose the policy-based method that resolves these problems. The proposed method does not completely rely on the settings of a client and can determine an applied security profile using a policy and the context of the client's request. Therefore, only one authorization server or isolated environment, such as a realm of an authorization server, is required to support multiple different security profiles. Additionally, the proposed method can implement a security profile as a pluggable software module. Thus, the source code of the authorization server need not be modified. The proposed method and Financial-grade application programming interface (FAPI) security profiles were implemented in Keycloak, which is an open-source identity and access management solution, and evaluation scenarios were executed. The results of the evaluation confirmed that the proposed method resolves these problems. The implementation has been contributed to Keycloak, making the proposed method and FAPI security profiles publicly available.

***key words:*** *OAuth 2.0, security profile, financial-grade API (FAPI) security profile, open banking*

## 1. Introduction

OAuth 2.0 is a widely used web-based authorization protocol [1] that is used in a wide range of use cases. One such use case is open banking, where a financial institution provides its financial services via application programming interfaces (APIs) and an application built by a third-party developer accesses them on behalf of a user of the financial institution.

OAuth 2.0 defines multiple protocol flows and flexibility in the usage is left. This flexibility, however, often introduces security holes if it is used incorrectly. For example, OAuth 2.0 recommends including a "state" parameter in an authorization request, which means that using OAuth 2.0 without the "state" parameter can be allowed. However, a cross-site request forgery attack might succeed if using OAuth 2.0 without the "state" parameter (Sect. 4.4.1.8 of [2]).

To prevent introduction of security vulnerabilities, detailed methods for using OAuth 2.0 securely have been developed, which are called *security profiles*. A security profile consists of the requirements that a client request to an authorization server must satisfy. Applying a security profile means that an authorization server judges if a client request satisfies requirements of the security profile. The authorization server returns a normal response if the request satisfies the requirements; otherwise, the authorization server returns an error response.

There is a method called the *settings-based method* to apply a security profile. This method can only apply one security profile to a client request because all requests from the same client are processed in the same way based on these settings.

The settings-based method utilizes settings of an authorization server and client. Some settings are defined by OAuth 2.0 related specifications [3], [4], while others are defined by an authorization server product. An authorization server manages its settings and those of the clients. When a client sends a request to an authorization server, it processes the request and returns its response based on the server and client settings. The setting-based method makes the authorization sever to set appropriate values for these settings to check if a client request satisfies the requirements of the security profile.

However, the settings-based method increases managerial costs for an authorization server administrator when an authorization server needs to support several security profiles and apply a different security profile to a different client request. The authorization server administrator needs to create and manage an authorization server per security profile because only one security profile can be applied using the settings-based method. If an authorization server supports multi-tenant capability, the authorization server administrator needs to create and manage a tenant for each security profile. The authorization server administrator also needs to register the same clients and their users in all authorization servers or tenants and manage these duplicated clients and users.

Further, regardless of the use case, if the existing settings are inadequate for applying a security profile, new settings and logic need to be added to an authorization server,

Manuscript received October 13, 2022.
Manuscript revised March 21, 2023.
Manuscript publicized June 20, 2023.
[†]The authors are with Hitachi, Ltd., Tokyo, 100–8280 Japan.
[††]The authors are with Okayama University, Okayama-shi, 700–8530 Japan.
a) E-mail: takashi.norimatsu.ws@hitachi.com
DOI: 10.1587/transinf.2022ICP0004

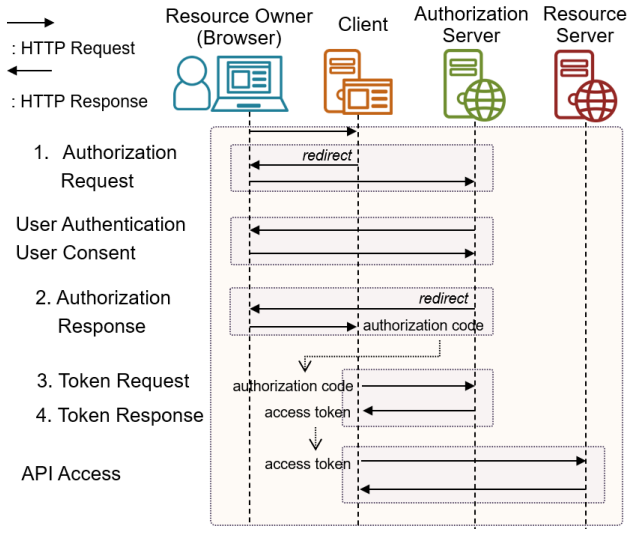


**Fig. 1** The authorization code flow of OAuth 2.0

which generally requires source code modifications. However, this may make it impossible to apply the security profile because source code cannot be modified if an authorization server administrator uses proprietary software of an authorization server. Conversely, the source code can be modified if the authorization server administrator uses open-source software. Moreover, modifying the source code may impair a server's functionalities, resulting in errors and quality degradation.

To avoid setting up and managing a lot of setting items for multiple purposes, a policy can be generally used. For example, a policy was used for access control with OAuth 2.0 in [5]. Some products of an authorization server used a policy for multiple purposes. Keycloak 15 [6], ForgeRock AM 7.1 [7], Okta Classic Engine 2022.03 [8], and Auth0 [9] uses a policy for access control on resource servers' resources. However, the policy used in them was not intended to be used for applying security profiles.

We aimed to propose a new method using a policy (the *policy-based method*) for applying a security profile to prevent these problems. The proposed policy-based method receives a client request and evaluates a policy to check if the request meets its conditions for applying a security profile. If the result of the evaluation is positive, the security profile is applied to the request.

The proposed policy-based method is applied to the authorization code flow of OAuth 2.0 that is utilized for use cases where a third-party application accesses APIs like open banking, as shown in Fig. 1.

When a third-party application (called a "client" in OAuth 2.0) accesses an API that operates a resource of a user (called a "resource owner" in OAuth 2.0) on behalf of the user, the application creates an authorization request and sends it to an authorization server by redirecting the user's browser ("1. Authorization Request" in Fig. 1).

After authenticating the user ("User Authentication" in Fig. 1) and obtaining the consent from the user for the application to access the API that operates the user's resource on behalf of the user ("User Consent" in Fig. 1), the authorization server generates an authorization code, creates an authorization response that includes the authorization code, and sends it to the application by redirecting the user's browser ("2. Authorization Response" in Fig. 1).

After receiving the authorization response, the application sends a token request including the authorization code to the authorization server ("3. Token Request" in Fig. 1).

After receiving the token request, the authorization server generates an access token proving that the user agrees to accessing the API that operates the user's resource by the application and sends a token response including the access token to the application ("4. Token Response" in Fig. 1).

After receiving the token response, the application accesses the API using the access token ("API Access" in Fig. 1).

The policy-based method can support multiple security profiles in a single authorization server or tenant. It does not completely rely on client settings and can determine which security profile is applied to a client request based on its content. Therefore, it can apply a different security profile to a different client request based on its content.

The policy-based method need not modify the authorization server's source code for applying a security profile. It allows an authorization server administrator to implement a security profile and a condition that determines if the security profile is applied or not in a pluggable manner.

Herein, the policy-based method was implemented in Keycloak and evaluated to prove that this method can prevent the aforementioned problems [10]. Keycloak is a widely used identity and access control open-source software for authentication and authorization purposes [11], [12].

By using the proposed method, some standardized security profiles were implemented without modifying Keycloak's body source code. Financial-grade API Security Profile 1.0 Baseline (FAPI1-baseline) [13], Financial-grade API Security Profile 1.0 Advanced (FAPI1-advanced) [14], and Financial-grade API Client Initiated Backchannel Authentication Profile (FAPI-CIBA) [15] were implemented. These security profiles were standardized by OpenID Foundation (OIDF) and used in several in-service open banking applications. For example, the security profiles of UK Open Banking [16], Australia Consumer Data Right [17], and Open Banking Brazil [18] use them as a basis for their security profiles.

A client was simulated to confirm that the method resolved the problem of increasing managerial costs for an authorization server administrator. Client requests were sent to Keycloak with scope parameter values that indicated different type of APIs the client needed for access. Additionally, it was confirmed that the policy-based method can change a security profile based on the scope parameter value. Therefore, only a single realm or authorization server was required to support multiple security profiles, which reduces the managerial costs for an authorization server administra-

tor.

In addition, Keycloak, along with the policy-based method for UK Open Banking and Open Banking Brazil, was used to estimate and clarify how the managerial costs for an authorization server administrator can be reduced using the proposed policy-based method.

As an additional objective of the policy-based method, it was implemented with product level quality that can be used in the real world. The proposed implementation was contributed to Keycloak, reviewed by Keycloak maintainers, and merged into the Keycloak upstream repository. Therefore, this proposed implementation is now publicly available for use.

The preliminary version of this study described the policy-based method for Keycloak only and clarified how the managerial costs for an authorization server administrator were reduced by using this method [19]. This study generalized the policy-based method for a general authorization server and analyzed the evaluation result, proving that the policy-based method can apply a different security profile to a different client request based on its content by using Keycloak.

## 2. Problems

Two problems were discovered if an authorization server was used to apply a security profile with the settings-based method in a use case that had the following three characteristics (such as open banking):

- Characteristic 1: an authorization server needs to support several security profiles and apply a different security profile to a different request from the same client for accessing an API that requires a different security level.

Financial institutions provide several types of APIs (e.g., account information and payment initiation in UK Open Banking) [20]. Each type of API requires a different security level depending on its service. Therefore, a different security profile is applied to a client request for accessing an API that requires a different security level.

UK Open Banking is a use case that supports several security profiles, including FAPI1-advanced and FAPI-CIBA.

Open Banking Brazil supports several security profiles and has also defined its own security profile called "Open Banking Brazil Financial-grade API Security Profile 1.0 (OB-BR FAPI)", which is based on FAPI1-advance. OB-BR FAPI optionally uses FAPI-CIBA. When authenticating a user, OB-BR FAPI requires an authentication method with a level of assurance of at least two if a client request wants to access Read-Only APIs. The security profile requires an authentication method with a level of assurance of at least three if a client request wants to access Read-Write APIs (Sect. 5.2.2.4 of [18]). Therefore, four variants of OB-BR FAPI (Read-Only APIs, Read-Write APIs, FAPI-CIBA + Read-Only APIs, FAPI-CIBA + Read-Write API) can be used in Open Banking Brazil.

- Characteristic 2: an authorization server needs to manage many clients.

UK Open Banking is a use case that needs to manage many clients. There are 231 third party providers (clients in OAuth 2.0) registered in UK Open Banking [21].

Open Banking Brazil is also a use case that needs to manage many clients. There are 94 data transmitters/recipients (clients in OAuth 2.0) [22] and 54 payment initiation service providers (clients in OAuth 2.0) [23] registered in Open Banking Brazil.

- Characteristic 3: an authorization server needs to support an authorization code flow (Fig. 1) and apply the same security profile to authorization and token requests that belong to the same authorization code flow.

UK Open Banking, Australia Consumer Data Right, and Open Banking Brazil are use cases that require an authorization code flow.

An authorization code flow consists of authorization and token requests/responses. Therefore, an authorization server needs to apply the same security profile to a token request that corresponds to the authorization request if it applies a security profile to an authorization request.

The following two problems occur if an authorization server applies a security profile using the settings-based method in a use case that has these three characteristics.

- Problem 1: the extent of operations increases for an authorization server to manage entities, such as clients and users.

An authorization server administrator needs to create and manage an authorization server per security profile because only one security profile can be applied.

Some authorization servers support a logically isolated environment. For example, Keycloak [24] and ForgeRock Access Management (AM) [25] call this a realm. Okta [26] and Auth0 [27] call it a tenant, and Amazon Cognito [28] calls it a user pool. In this paper, the term "realm" is used to represent the environment. A realm is fully isolated from other realms and has its own configuration, set of clients, and their users. This allows a single installation of an authorization server to be used for multiple purposes.

Nevertheless, multiple authorization servers or realms need to be created and managed, and the duplicated clients and users need to be registered and managed in all authorization servers or realms. If the client or user settings are modified in one authorization server or realm, then they need to be modified in other authorization servers or realms. However, this increases managerial costs for an authorization server administrator.

Generally, an authorization server product has many client settings. For example, Keycloak 15 has 110 [29], ForgeRock AM 7.1 has 65 [30], PingFederate 11 has 55 [31], Auth0 has 44 [32], and Okta Classic Engine 2022.03 has 19 [33]. If Keycloak 15 is used in UK Open

Banking, the number of managed client settings would be 25,410 (number of clients (231) × number of client settings (110)). Similarly, if Keycloak 15 is used in Open Banking Brazil, the number of managed client settings would be 15,950 (number of clients (145) × number of client settings (110)).

If Keycloak 15 is used in UK Open Banking, the number of managed client settings would be 50,820 (number of security profiles (2 variants, FAPI1-advanced and FAPI-CIBA) × number of clients (231) × number of client settings (110)). Similarly, if Keycloak 15 is used in Open Banking Brazil, the number of managed client settings would be 63,800 (number of security profiles (4 variants of OB-BR) × number of clients (145) × number of client settings (110)). In addition, the increase in number of operations increases the risk of operational mistakes, which often can cause security incidents.

- Problem 2: an authorization server's source code needs to be modified to add new settings and logic to apply the security profile if existing settings are inadequate.

Modifying an authorization server's source code is generally impossible if it uses proprietary software. An authorization server administrator can try to modify the source code if open-source software but this may impair server functionalities and cause errors and quality degradation.

## 3. Policy-Based Method for Applying a Security Profile

A new policy-based method was designed to resolve the problems that arise if the settings-based method is used for applying a security profile.

### 3.1 Design Principles

The design principles of the policy-based method are as the following four:

- Design principle 1: logic about security profiles can be independent of client settings to remove the necessity of configuring every client to apply a security profile.

This design principle contributes to resolving problem 1. The extent of managerial operations for clients decreases upon removing the necessity of configuring every client for applying a security profile.

- Design principle 2: a security profile applied to a client request can be changed dynamically to support multiple security profiles in a single realm.

This design principle contributes to resolving problem 1. An authorization server only needs to manage one realm by supporting multiple security profiles in a single realm. Therefore, the extent of managerial operations decreases for entities included in the realm, such as clients and their users.

- Design principle 3: the policy-based method consists of the following two separated parts, which can be implemented as pluggable software modules:
  - Policy: holding a reference to a security profile and determining if the security profile is applied.
  - Profile: applying a security profile.

This design principle contributes to resolving problem 2, allowing an authorization server administrator to implement a security profile without modifying an authorization server's source code.

- Design principle 4: the policy-based method can recognize which authorization and token requests belong to the same authorization code flow.

This design principle considers characteristic 3.

### 3.2 Logical Components

The proposed method applied attribute based access control (ABAC) because all entities that it interacts with can be represented as entities of ABAC. As you can see in Fig. 2, a client, authorization and token endpoints, and the proposed method can be represented as a subject, policy enforcement points (PEP), and policy decision point (PDP) in ABAC respectively.

The policy-based method was designed to include six logical components according to the design principles: the handler, flow context, policy, condition, profile, and executor, as shown in Fig. 2.

A *handler* is a component that receives an authorization request ("1. Authorization Request" of Fig. 1), evaluates policies of the policy-based method, applies profiles of the policy-based method, and saves the context of the request to the flow context. It also receives a token request ("3. Token Request" of Fig. 1), retrieves the context of the authorization request, evaluates policies of the policy-based method, and applies profiles of the policy-based method. The handler follows design principle 4.

A *flow context* is a component that holds the context of an authorization request. The context can be retrieved by an authorization code ("2. Authorization Response" of Fig. 1) included in a token request. Therefore, an authorization request is bound with a token request by an authorization code. When an authorization server receives a token request with an authorization code, the context of its corresponding authorization request can be retrieved using the authorization code. The flow context follows design principle 4.

A *condition* is a component that includes part of the logic to determine if a profile of the policy-based method applies to a client request. By following design principle 2, the condition can use the context data of the request (e.g., the parameters included in the request and HTTP headers) to determine if a profile of the policy-based method can be applied to the request. Therefore, it can change the security profile based on the content of the request. There are three possible results from the condition evaluation: Vote.YES, Vote.NO, and Vote.ABSTAIN. Vote.YES indicates that a profile of the policy-based method was applied to a client request. Vote.NO indicates that a profile of the policy-based

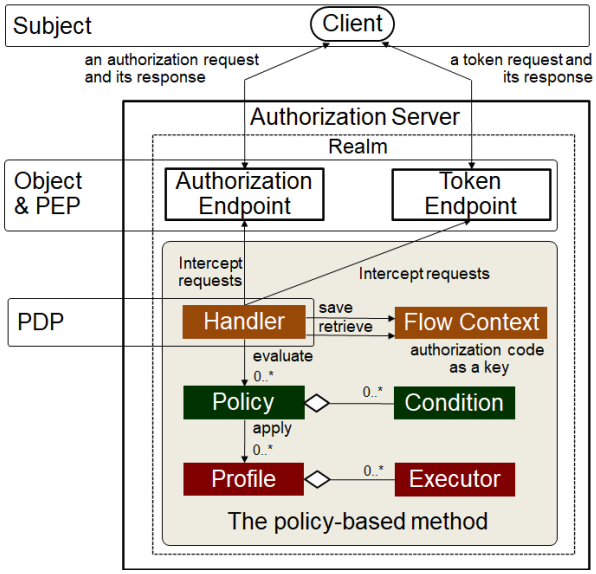


**Fig. 2** Class diagram of components of the policy-based method

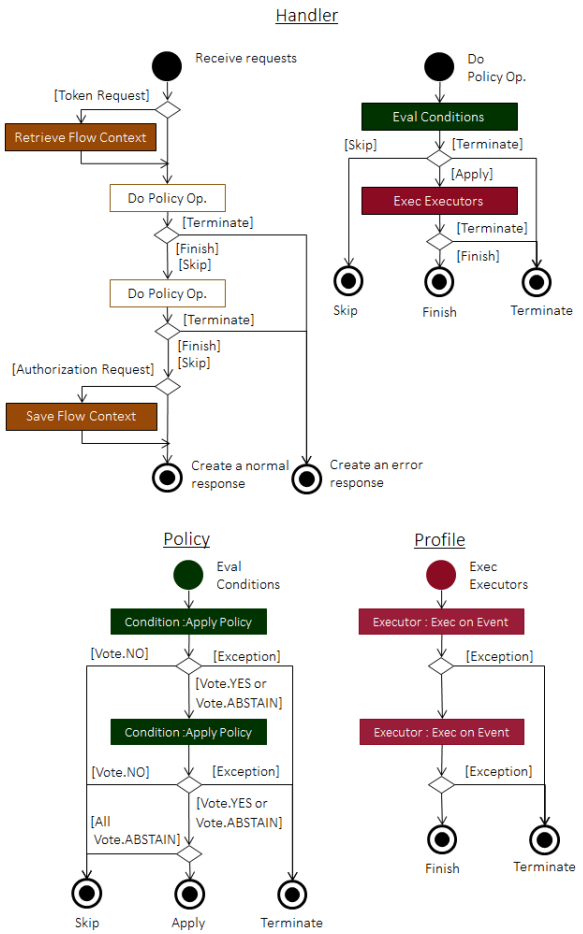


**Fig. 3** Activity diagram of the policy-based method

method was not applied to a client request. Vote.ABSTAIN indicates that a condition's evaluation was skipped. By following design principle 3, the condition is implemented as pluggable software module.

A *policy* is a component that includes all logic needed to determine if a profile of the policy-based method is applied to a client request. A policy of the policy-based method consists of multiple conditions. The policy of the policy-based method applies a security profile to the request if the evaluation result of the condition does not include Vote.NO and includes at least one Vote.YES and zero or several Vote.ABSTAIN. The policy of the policy-based method follows design principle 3.

An *executor* is a component that includes the logic needed to apply part of a security profile. Generally, an executor ascertains if a request satisfies one of the requirements of a security profile. The process of the policy-based method is terminated if the request does not satisfy the executor. The logic does not completely depend on client settings by following design principle 1. The executor is implemented as a pluggable software module by following design principle 3.

A *profile* is a component that includes the logic needed to apply a security profile. A profile of the policy-based method consists of multiple executors. Generally, a profile of the policy-based method ascertains if a client request satisfies all requirements of a security profile. If the request does not satisfy all the requirements, the process of the policy-based method is terminated and an error response is returned to the client. The profile of the policy-based method follows design principle 3.

For example, the situation is considered where two profiles of the policy-based method include two executors, and two policies of the policy-based method include two conditions and one reference to a profile of the policy-based method, as shown in Fig. 3.

The handler sequentially evaluates two policies of the policy-based method when it receives an authorization request from a client. The handler applies a profile of the policy-based method if the results of the condition evaluation do not include Vote.NO, and include at least one Vote.YES and zero or several Vote.ABSTAIN. The handler then implements executors of the profile of the policy-based method. The handler saves a context of the authorization request using an authorization code as a key if all the executors' executions are successfully completed. This key is used to retrieve the saved context later and returns a normal response to the client.

The handler retrieves the context of the authorization request using an authorization code in the token request as a key when the handler receives a token request from the client. Then, the handler performs the same process as when it received the authorization request. The handler evaluates two policies of the policy-based method sequentially by using the context of the authorization request and receives the evaluation results the same way it received the authorization request. Therefore, the handler executes the same profile of the policy-based method as when it received the authorization request, which means that the handler applies the same

security profile to the authorization and token requests that belong to the same authorization code flow.

## 4. Implementation

As mentioned earlier, the policy-based method and security profiles were implemented in Keycloak.

### 4.1 Objective

An additional objective of this study was to show that the implementation had product level quality and can be used in the real world. To achieve this objective, the proposed implementation would need to be accepted by Keycloak's maintainers and merged into the Keycloak upstream repository.

Adding our implementation would increase the size of Keycloak's body source code, which is not preferable from the perspective of Keycloak's maintainers. This is because the maintenance cost of Keycloak would increase, which would make it difficult to maintain the high quality survive provided by Keycloak. Therefore, the policy-based method needed to be implemented considering this aspect.

### 4.2 Implementation of the Policy-Based Method

A profile and policy of the policy-based method were implemented as independent software components by following design principle 3. If the profile and policy of the policy-based method were combined into one software component, the profile of the policy-based method would be duplicated if multiple policies of the policy-based method referred the same profile of the policy-based method, which would lead to redundant source code.

A condition and executor can be implemented as a plug-in (called a provider) by following design principle 3, which is achieved using a service provider interface (SPI) defined by Java's language specification. A factory pattern was used to instantiate the provider, as shown in Fig. 4. If the whole policy-based method was implemented in Keycloak, new source code would need to be added to Keycloak's body source code when a new security profile is supported by Keycloak. Therefore, the size of Keycloak's body source code will increase whenever a new security profile is supported by Keycloak.

A new security profile can be supported for SPI by implementing executors and conditions as plug-ins. Therefore, new source code need not be added to Keycloak's body source code when a new security profile is supported by Keycloak. Additionally, the size of Keycloak's body source code will not increase whenever a new security profile is supported by Keycloak.

The policy-based method does not depend on any specifications or features of a specific authorization server product. Therefore, an authorization server product other than Keycloak can implement the policy-based method. According to design principle 3, conditions and executors should

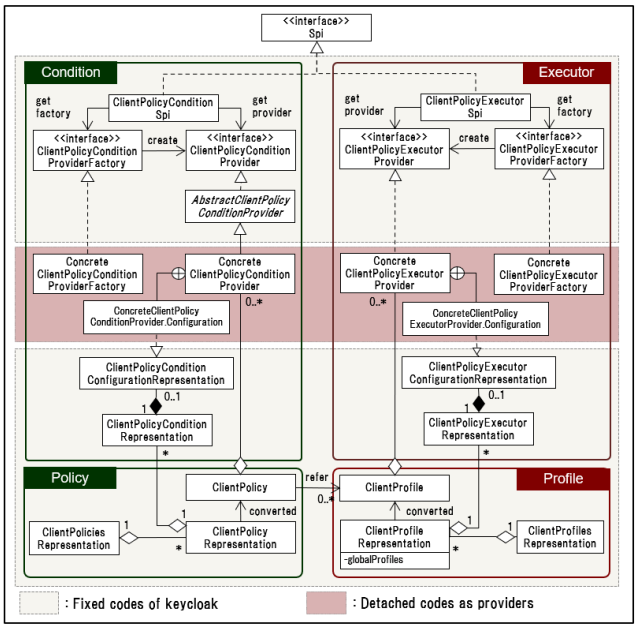


**Fig. 4** Class diagram of implementation of the policy-based method

be pluggable. SPI can be used if an authorization server product is implemented in Java.

### 4.3 Implementation of Security Profiles

The policy-based method implemented executors used by three security profiles: "fapi-1-baseline" for FAPI1-baseline (Table 1), "fapi-1-advanced" for FAPI1-advanced (Table 2), and "fapi-ciba" for FAPI-CIBA (Table 3). Subsequently, the three security profiles were implemented using the executors, which showed that a security profile can be implemented in a pluggable manner. Therefore, the policy-based method can resolve problem 2.

Finally, several conditions were implemented that can be used to determine which security profile was applied to a client request (Table 4). One of the implemented conditions was "client-scopes", which checks the OAuth 2.0's scope parameter value of a client request. The scope parameter can be used to determine the type of API that a client wants to access. Therefore, this "client-scopes" condition was used in the evaluation.

### 4.4 Implementation of User Interfaces (UIs) for Administering the Policy-Based Method

The proposed policy-based method has not only its APIs, command line interfaces (CLIs) but also UIs for a system administrator. By using the UIs, the system administrator can apply the policy-based method to the actual cases in real environment with less costs compared with using the APIs or CLIs directly.

There are two types of UIs: one for profiles of the policy-based method and the other for policies of the policy-based method. As a default, the UI for profiles of

**Table 1** Executors of the profile of the policy-based method ”fapi-1-baseline” for FAPI1-baseline

| Type | Description |
|---|---|
| secure-session | Requires a client to include ”nonce” parameter defined in OIDC or ”state” parameter defined in OAuth 2.0 in its request. |
| pkce-enforcer | Requires a client to follow Proof Key for Code Exchange [29] operation with secure algorithm, such as S256. |
| secure-client-authenticator | Requires a client to use a secure client authentication method. |
| secure-client-uris | Requires a client to register universal resource identifiers (URIs) in a secure manner. |
| consent-required | Requires a client to obtain consent from a user to access an API. |
| full-scope-disabled | Requires a client to use only pre-registered values for ”scope” parameter. |

**Table 2** Executors of the profile of the policy-based method ”fapi-1-advanced” for FAPI1-advanced

| Type | Description |
|---|---|
| secure-session | Requires a client to include ”nonce” parameter defined in OIDC or ”state” parameter defined in OAuth 2.0 in its request. |
| secure-client-authenticator | Requires a client to use a secure client authentication method. |
| secure-client-uris | Requires a client to register universal resource identifiers (URIs) in a secure manner. |
| consent-required | Requires a client to obtain consent from a user to access an API. |
| full-scope-disabled | Requires a client to use only pre-registered values for ”scope” parameter. |
| confidential-client | Requires a client to be a confidential client defined in OAuth 2.0. |
| secure-request-object | Requires a client to use a request object in its authorization request. |
| secure-response-type | Requires a client to send its authorization request with code id_token or code id_token token in its response type. |
| secure-signature-algorithm | Requires a client to use secure signature algorithms. |
| secure-signature-algorithm-signed-jwt | Requires a client to use secure signature algorithms for the client authentication using JWT. |
| holder-of-key-enforcer | Requires a client to use a Holder-of-Key token instead of a bearer token. |

**Table 3** Executors of the profile of the policy-based method ”fapi-ciba” for FAPI-CIBA

| Type | Description |
|---|---|
| secure-ciba-req-sig-algorithmn | Requires a client to send a signed authentication request. |
| secure-ciba-session | Requires a client to include ”binding_message” in its request. |
| secure-ciba-signed-authn-req | Requires a client to use secure signature algorithms. |

**Table 4** Conditions for a policy of the policy-based method

| Type | Description |
|---|---|
| client-scopes | Checks if a client request includes a ”scope” parameter whose value includes a pre-registered value. |
| client-roles | Checks if a pre-registered role is assigned to a client. |
| client-access-type | Checks if a client is a pre-registered type defined in OAuth 2.0 (confidential, public, bearer-only). |
| any-client | Accepts any client request. |

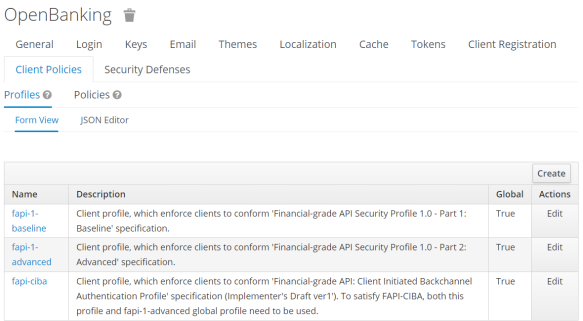


**Fig. 5** Keycloak’s UI for profiles of the policy-based method

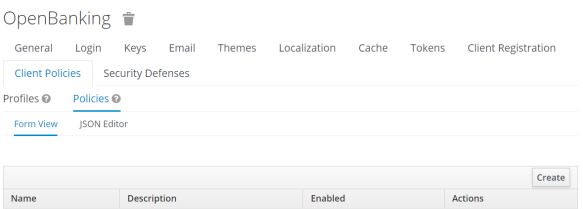


**Fig. 6** Keycloak’s UI for policies of the policy-based method

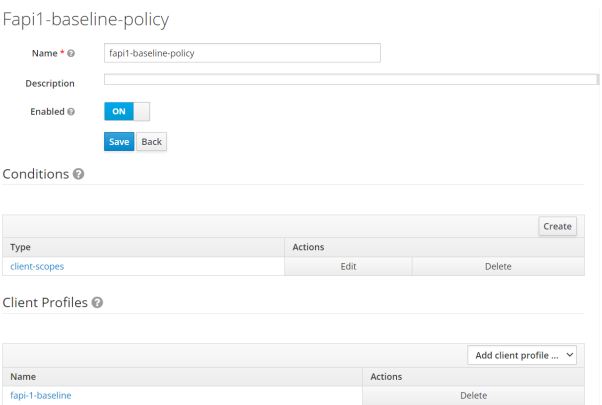


**Fig. 7** Keycloak’s UI for policy of the policy-based method

the policy-based method (Fig. 5) shows three profiles of the policy-based method: FAPI1-baseline (fapi-1-baseline), FAPI1-advanced (fapi-1-advance), and FAPI-CIBA (fapi-ciba). These are built-in profiles of the policy-based method; therefore, authorization server administrator does not need to create profiles of the policy-based method by themselves.

As a default, the UI for the policies of the policy-based method (Fig. 6) shows no policies of the policy-based method; therefore, an authorization server administrator needs to create policies of the policy-based method by their own. However, conditions in Table 4 have been created in advance; therefore, an authorization server administrator can use these conditions to create their own policies of the policy-based method. For example, if an authorization administrator want to apply FAPI1-baseline to a client request

with OAuth 2.0's scope parameter including a specific value, they only need to create a policy including client-scopes condition in Table 4 on the UI for policy of the policy-based method (Fig. 7).

By using the built-in profiles of the policy-based method and conditions, an authorization server administrator can setup and apply the policy-based method to a system in real environments on the UIs for profiles and policies of the policy-based method.

## 5. Evaluation

As mentioned in Sect. 4.3, the implemented policy-based method can resolve problem 2. Two evaluation scenarios were executed to confirm that the implemented policy-based method considered the three characteristics of the open banking use case and resolved problem 1. Then, the managerial costs were estimated for clients when Keycloak was used in open banking use cases to clarify how managerial costs for an authorization server administrator could be reduced with the policy-based method.

The first scenario was used to confirm that the implemented policy-based method can apply the same security profile to authorization and token requests that belong the same authorization code flow, which means that the policy-based method considered characteristic 3. The second scenario was used to confirm that the implemented policy-based method can apply a different security profile to a unique client request based on the request's context, which means that the policy-based method considered characteristics 1 and 2 and resolved problem 1. Finally, the managerial costs were estimated for clients if Keycloak 15 was used for UK Open Banking and Open Banking Brazil.

### 5.1 Setup

The proposed implementation was accepted and merged into Keycloak 15.

Keycloak can run on Windows and Linux operating systems using a Java virtual machine. In this study, Keycloak was run on Windows (Windows 10 Pro edition, 21H2 version, 19044.1706 build) with OpenJDK (11.0.14.1 version, Microsoft-31205 build, 11.0.14.1+1-LTS runtime environment). Keycloak supports two modes: production and development. Even though our implementation can work on both modes, Keycloak was run in development mode because the production mode required many strict settings that were not needed for the evaluation.

Before executing evaluation scenarios, the following setup was performed:

1. Creating a realm.
   - Realm name: OpenBanking
2. Adding a user.
   - Username: john
3. Adding scopes for two different types of APIs that a client accesses.
   - Scope name: read_account_api
   - Scope name: bank_transfer_api
4. Adding policies of the policy-based method for two different types of APIs that a client accesses using JavaScript Object Notation (JSON), as shown in Fig. 8.

   This JSON representation included a policy of the policy-based method named "fapi-1-baseline-policy" (lines 2-10), whose evaluation is positive only if the OAuth 2.0's scope parameter value of a request includes "read_account_api" (lines 5-9). It then applies a profile of the policy-based method for the FAPI1-baseline named "fapi-1-baseline" (line 10), a policy of the policy-based method named "fapi-1-advanced-policy" (lines 11-19), whose evaluation is positive only if the OAuth 2.0's scope parameter value of a request includes "bank_transfer_api" (lines 14-18), and a profile of the policy-based method for FAPI1-advanced named "fapi-1-advanced" (line 19).
5. Adding a client with the following settings.
   - Client ID: fintech-app
   - Client Protocol: openid-connect
   - Consent Required: ON
   - Access Type: Confidential
   - Valid Redirect URIs: https://fintech-app.example.com/cb
   - Optional Client Scopes - Available Client Scopes: read_account_api, bank_transfer_api

   Keycloak uses these settings to recognize that the "fintech-app" client may send an authorization request with these scope parameter values.

```
1:{"policies": [{
2:  "name": "fapi1-baseline-policy",
3:  "enabled": true,
4:  "conditions": [{
5:    "condition": "client-scopes",
6:    "configuration": {
7:      "is-negative-logic": false,
8:      "scopes": ["read_account_api"],
9:      "type": "Optional"}}],
10:   "profiles": ["fapi-1-baseline"]}, {
11: "name": "fapi1-advanced-policy",
12: "enabled": true,
13: "conditions": [{
14:   "condition": "client-scopes",
15:   "configuration": {
16:     "is-negative-logic": false,
17:     "scopes": ["bank_transfer_api"],
18:     "type": "Optional"}}],
19:   "profiles": ["fapi-1-advanced"]}]}
```

**Fig. 8** JSON representations of policies of the policy-based method

### 5.2 Executing Evaluation Scenarios

The following five types of evidence were used for the evaluation to confirm that the policy-based method worked as expected: an authorization request, authorization response, token request, token response, and Keycloak's log.

Contents and formats of an authorization request, authorization response, token request, and token response were defined using OAuth 2.0's specifications. Authorization and token responses have two types of response: normal and error.

The Keycloak log contains three types of log entries about the policy-based method: the policy of the policy-based method, condition, and profile/executor of the policy-based method. A log entry about policy of the policy-based method has two types of sub log entries: starting evaluation of a policy of the policy-based method (labeled as POLICY OPERATION) and finishing evaluation of a policy of the policy-based method. The finishing evaluation shows two different results of the evaluation: applying a policy of the policy-based method (labeled as POLICY APPLIED) and not applying a policy of the policy-based method (labeled as POLICY UNSATISFIED). A log entry about condition has two types of sub log entries: the starting evaluation of a condition (labeled as CONDITION OPERATION) and finishing evaluation of a condition. The finishing evaluation shows two different results of the evaluation: satisfying a condition (labeled as CONDITION SATISFIED) and not satisfying a condition (labeled as CONDITION NEGATIVE). Log entries about a profile and executor of the policy-based method were combined into one log entry, which included the type of executor and name of the profile of the policy-based method to which it belonged. The log entry about a profile and executor of the policy-based method has two types of sub log entries: the starting execution of an executor (labeled as EXECUTOR) and finishing execution of an executor. The finishing execution was not outputted if the execution of the executor was successfully completed. If the execution of the executor fails, it is outputted with an EXECUTOR EXCEPTION label, which means that a client request did not satisfy a requirement of the security profile. Other log entries for debugging were outputted other than these log entries.

The Chrome browser developer tool was used to send an authorization request and receive an authorization response. cURL was used to send a token request and response.

### 5.2.1 Scenario 1: Applying the Same Security Profile to Authorization and Token Requests of the Same Authorization Code Flow

The aim of this scenario is to confirm that the policy-based method considers characteristic 3 of the open banking use case. To achieve the aim, it was determined if the policy-based method could apply the same security profile to authorization and token requests of the same authorization code flow by executing the evaluation scenario shown in Fig. 9.

First, the following authorization request was sent, which satisfies all requirements of the FAPI1-baseline:

```
http://localhost:8080
/realms/OpenBanking/protocol/openid-connect/auth?
client_id=fintech-app&
```

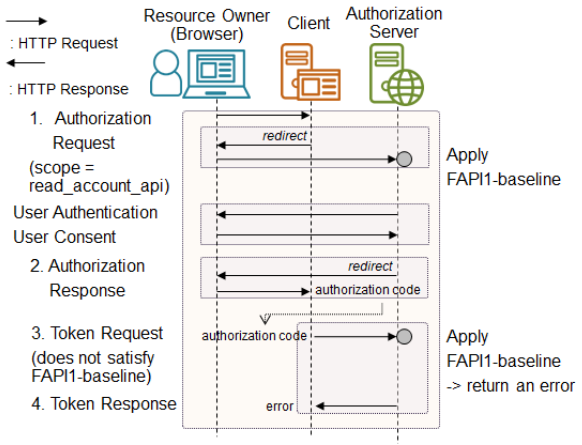


**Fig. 9** Evaluation scenario 1 and its expected result

```
redirect_uri=https://fintech-app.example.com/cb&
state=a8159cbf-2e98-4438-803c-f52acb1b6d6e&
response_type=code&
scope=read_account_api&
code_challenge_method=S256&
code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
```

This request included the "read_account_api" scope parameter value.

It was expected that the authorization code would be successfully returned by applying the FAPI1-baseline. If Keycloak operated as expected, it would return a normal response and output the following log entries when it receives an authorization request:

1. A policy of the policy-based method named "fapi1-baseline-policy" would be evaluated (labeled as POLICY OPERATION) and applied as the result of the evaluation (labeled as POLICY APPLIED).
2. A profile of the policy-based method named "fapi-1-baseline" would be executed and no log entry labeled as EXECUTOR EXCEPTION would be outputted, which indicates that the client request satisfied all requirements of the FAPI1-baseline.
3. A policy of the policy-based method named "fapi1-advanced-policy" would be evaluated (labeled as POLICY OPERATION) and not applied as the result of the evaluation (labeled as POLICY UNSATISFIED).

Upon receiving the request, Keycloak outputted the following log entries on the console, which have been redacted for readability:

```
1:2022-03-26 12:54:13,396 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY OPERATION ::
  policy name = fapi-1-baseline-policy
... skip
2:2022-03-26 12:54:13,404 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY APPLIED ::
  policy name = fapi-1-baseline-policy
... skip
3:2022-03-26 12:54:13,426 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) EXECUTION ::
  policy name = fapi-1-baseline-policy,
  profile name = fapi-1-baseline, executor name = class
  org.keycloak.services.clientpolicy.executor.
```

```
  SecureSessionEnforceExecutor,
  provider id = secure-session
... skip
4:2022-03-26 12:54:13,431 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY OPERATION ::
  policy name = fapi-1-advanced-policy
... skip
5:2022-03-26 12:54:13,443 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY UNSATISFIED ::
  policy name = fapi-1-advanced-policy
```

Line 1 shows that Keycloak started evaluating the policy of the policy-based method for the FAPI1-baseline named "fapi-1-baseline-policy" and created step 4 of Sect. 5.1. Line 2 shows that Keycloak concluded that the policy of the policy-based method evaluation for FAPI1-baseline was positive. Therefore, Keycloak applied the FAPI1-baseline to the request. Lines 1 and 2 of the log entries matched number 1 of the expected log entries.

Line 3 shows that Keycloak started applying the FAPI1-baseline to the authorization request. Subsequently, no log entries labeled as EXECUTOR EXCEPTION were outputted when applying the FAPI1-baseline. Therefore, the log entry matched number 2 of the expected log entries.

Line 4 shows that Keycloak started evaluating the policy of the policy-based method for the FAPI1-advanced named "fapi-1-advanced-policy" created in step 4 of Sect. 5.1. Line 5 shows that Keycloak concluded that the policy of the policy-based method evaluation for FAPI1-advanced was negative. Therefore, Keycloak did not apply FAPI1-advanced to the request. Lines 4 and 5 of the log entries matched number 3 of the expected log entries.

According to the log entities, Keycloak operated as expected when it received the authorization request.

Next, Keycloak returned a page requiring user authentication and authorization, which is covered by the user added in step 2 of Sect. 5.1. Subsequently, the following successful authorization response was received as expected:

```
302 Redirect
https://fintech-app.example.com/cb?
state=a8159cbf-2e98-4438-803c-f52acb1b6d6e&
session_state=d9e45ee8-e772-4c3f-9c17-4f92edc3b650&
code=ac9c62f5-2def-457f-9fdc-31c4722c8eca.12913f04-300b-4bb8
-a766-f2532c83ab73.6f3908b8-17f6-4281-89f7-28efaaae689e
```

The following token request does not satisfy the FAPI1-baseline because it used the "client_secret_basic" client authentication method defined in Sect. 9 of OpenID Connect specification [35], which is not allowed by the FAPI1-baseline requirement 4 in Sect. 5.2.2 of [13]:

```
curl --location --request POST 'http://localhost:8080/realms/
OpenBanking/protocol/openid-connect/token'
--header 'Authorization: Basic ZmludGVjaC1hcHA6dWVPUExrdkc0R3lua
GFiS0pVcE1FcG95REUzSE14REI='
--header 'Content-Type: application/x-www-form-urlencoded'
--data-urlencode 'code=ac9c62f5-2def-457f-9fdc-31c4722c8eca.
12913f04-300b-4bb8-a766-f2532c83ab73.6f3908b8-17f6-4281-89f7-
28efaaae689e'
--data-urlencode 'grant_type=authorization_code'
--data-urlencode 'client_id=fintech-app'
--data-urlencode 'redirect_uri=https://fintech-app.example.com/cb'
--data-urlencode 'code_verifier=dBjftJeZ4CVP-mB92K27uhbUJU1p1r_
wW1gFWFOEjXk'
```

An "Authorization" header was calculated using a client secret that was generated randomly when the client was added to Keycloak. Therefore, the header value needed to be re-calculated to reproduce the evaluation.

A code parameter value, which is randomly generated by Keycloak per authorization request, must be included in the authorization response. Therefore, the code parameter value in an authorization response needs to be used for a token request to reproduce the evaluation.

The error response was expected to be returned by applying the FAPI1-baseline. If Keycloak operates as expected, it would return an error response and output the following log entries when it receives an authorization request:

4. A policy of the policy-based method named "fapi1-baseline-policy" would be evaluated (labeled as POLICY OPERATION) and applied as the result of the evaluation (labeled as POLICY APPLIED).
5. A profile of the policy-based method named "fapi-1-baseline" would be executed and a log entry (labeled as EXECUTOR EXCEPTION) would be outputted when an executor named "secure-client-authenticator" (Table 1) was executed, which means that a client request did not satisfy all requirements of the FAPI1-baseline.

Keycloak outputted the following log entries on the console after receiving the request, which have been redacted for readability:

```
1:2022-03-26 12:55:13,314 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-1) POLICY OPERATION ::
  policy name = fapi-1-baseline-policy
... skip
2:2022-03-26 12:55:13,320 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-1) POLICY APPLIED ::
  policy name = fapi-1-baseline-policy
... Skip
3:2022-03-26 12:55:13,324 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-1) EXECUTOR EXCEPTION :: executor name = class
  org.keycloak.services.clientpolicy.executor.
  SecureClientAuthenticatorExecutor,
  provider id = secure-client-authenticator,
  error = invalid_request,
  error detail = Configured client authentication method not
  allowed for client
```

Line 1 shows that Keycloak started evaluating the policy of the policy-based method for the FAPI1-baseline named "fapi-1-baseline-policy" that was created in step 4 of Sect. 5.1. Line 2 shows that Keycloak concluded that the policy of the policy-based method evaluation for the FAPI1-baseline was positive. Therefore, Keycloak applied the FAPI1-baseline to the request. Lines 1 and 2 of the log entries matched number 4 of the expected log entries.

Line 3 shows that Keycloak concluded that the token request did not satisfy all the FAPI1-baseline requirements when executing the executor named "secure-client-authenticator" (Table 1). Thus, Keycloak returned an error response. The log entry matched number 5 of the expected log entries.

According to the log entities, Keycloak operated as expected when it received the token request.

Finally, an error was received as a token response including its reason, as expected.

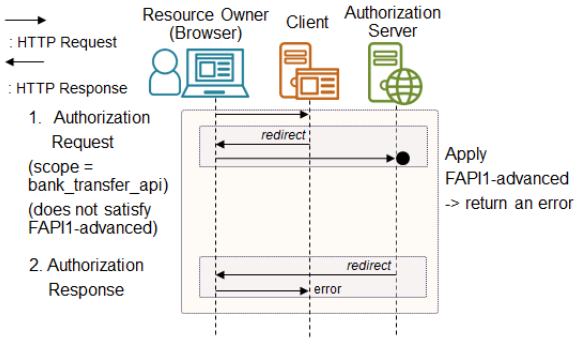


**Fig. 10** Evaluation scenario 2 and its expected result

```
400 Bad Request
{ "error": "invalid_grant",
  "error_description": "Configured client authentication method
   not allowed for client" }
```

According to the responses and logs, the policy-based method applied the same security profile (FAPI1-baseline) to the authorization and token requests of the same authorization flow as expected by following design principle 4. This indicates that the policy-based method considered characteristic 3 of the open banking use case.

### 5.2.2 Scenario 2: Applying a Different Security Profile Based on the Scope Parameter Value that Indicates a Type of API

The aim of this scenario is to confirm that the policy-based method can resolve problem 1 because only a single realm is needed to support multiple realms. To achieve the aim, it was determined if the policy-based method could be applied to a different security profile for an authorization request that included a different scope parameter value. This was done by executing an evaluation scenario to prove that the policy-based method supported multiple security profiles in a single realm, as shown in Fig. 10.

First, the following authorization request was sent that did not satisfy the FAPI1-advanced security profile because it did not include ether the ”request” or ”request_uri” parameter, which must be included in the request according to the FAPI1-advanced requirement 1 in Sect. 5.2.2 of [14].

```
http://localhost:8080
/realms/OpenBanking/protocol/openid-connect/auth?
client_id=fintech-app&
redirect_uri=https://fintech-app.example.com/cb&
state=a8159cbf-2e98-4438-803c-f52acb1b6d6e&
response_type=code&
scope=bank_transfer_api&
code_challenge_method=S256&
code_challenge=E9Melhoa2OwvFrEMTJguCHaoeK1t8URWbuGJSstw-cM
```

The request included the ”bank_transfer_api” scope parameter value.

An error response was expected to be returned by applying FAPI1-advanced. If Keycloak operated as expected, it would return an error response and output the following log entries when it receives an authorization request:

1. A policy of the policy-based method named ”fapi1-baseline-policy” would be evaluated (labeled as POLICY OPERATION) and not applied as the result of the evaluation (labeled as POLICY UNSATISFIED).
2. A policy of the policy-based method named ”fapi1-advanced-policy” would be evaluated (labeled as POLICY OPERATION) and applied as the result of the evaluation (labeled as POLICY APPLIED).
3. A profile of the policy-based method named ”fapi-1-advanced” would be executed and a log entry (labeled as EXECUTOR EXCEPTION) would be outputted when an executor named ”secure-request-object” (Table 2) was executed, which means that the client request did not satisfy all requirements of FAPI1-advanced.

On receiving the request, Keycloak outputted the following log entries on the console, which have been redacted for readability:

```
1:2022-03-26 14:16:03,414 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY OPERATION ::
  policy name = fapi-1-baseline-policy
... skip
2:2022-03-26 14:16:03,431 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY UNSATISFIED ::
  policy name = fapi-1-baseline-policy
... Skip
3:2022-03-26 14:16:03,431 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY OPERATION ::
  policy name = fapi-1-advanced-policy
... skip
4:2022-03-26 14:16:03,435 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) POLICY APPLIED ::
  policy name = fapi-1-advanced-policy
... Skip
5:2022-03-26 14:16:03,479 TRACE
  [org.keycloak.services.clientpolicy.DefaultClientPolicyManager]
  (executor-thread-0) EXECUTOR EXCEPTION :: executor name = class
  org.keycloak.services.clientpolicy.executor.
  SecureRequestObjectExecutor,
  provider id = secure-request-object, error = invalid_request,
  error detail = Missing parameter: 'request' or 'request_uri'
```

Line 1 shows that Keycloak started evaluating the policy of the policy-based method for the FAPI1-baseline ”fapi-1-baseline-policy” created in step 4 of Sect. 5.1. Line 2 shows that Keycloak concluded that the policy of the policy-based method evaluation for the FAPI1-baseline was negative. Therefore, Keycloak did not apply the FAPI1-baseline to the request. Lines 1 and 2 of the log entries matched number 1 of the expected log entries.

Line 3 shows that Keycloak started evaluating the policy of the policy-based method for the FAPI1-baseline named ”fapi-1-advanced-policy” created in step 4 of Sect. 5.1. Line 4 shows that Keycloak concluded that the policy of the policy-based method evaluation for FAPI1-advanced was positive. Therefore, Keycloak applied FAPI1-advanced to the request. Lines 3 and 4 of the log entries matched number 2 of the expected log entries.

Line 5 shows that Keycloak concluded that the token request did not satisfy all the FAPI1-advanced requirements when executing the executor named ”secure-request-object” (Table 2). Therefore, Keycloak returned an error response. The log entry matched number 3 of the expected log entries.

According to the log entities, Keycloak operated as expected when it received the authorization request.

Finally, an error was received as an authorization response including its reason, as expected.

```
302 Redirect
https://fintech-app.example.com/cb?
error=invalid_request&
error_description=Missing+parameter^%^3A+^%^27request^%^27+or+
^%^27request_uri^%^27&state=a8159cbf-2e98-4438-803c-f52acb1b6d6e
```

According to the responses and logs, the policy-based method applied the different security profile (FAPI1-advanced) to the request based on the scope parameter value (”bank_transfer_api”) in a single realm, as expected by following design principles 1 and 2. This indicates that the policy-based method can resolve problem 1 because only a single realm was needed to support multiple realms.

### 5.3 Estimating Managerial Costs

Keycloak 15 was used for UK Open Banking and Open Banking Brazil to estimate and clarify how managerial costs for an authorization server administrator can be reduced using the policy-based method that resolves problem 1.

If Keycloak is initialized as an authorization server for UK Open Banking, the number of managed client settings using the settings-based method would be 50,820 (number of security profiles (2) × number of clients (231) × number of client settings (110)). The number of managed client settings using the policy-based method is 25,410 (number of clients (231) × number of client settings (110)) because the policy-based method can support multiple security profiles in a single realm.

If a new security profile is added, the number of managed client settings for the settings-based method would be 25,410 (number of clients (231) × number of client settings (110)) because a new realm would need to be created for the new security profile. Conversely, the number of managed client settings would be 0 when using the policy-based method.

If the supported security profile is modified, the number of managed client settings using the settings-based method would be 25,410 (number of clients (231) × number of client settings (110)) because every client would need to be re-configured for the modified security profile. Conversely, the number of managed client settings would be 0 using the policy-based method.

If Keycloak is initialized as an authorization server for Open Banking Brazil, the number of managed client settings for the settings-based method would be 67,760 (number of security profiles (4) × number of clients (154) × number of client settings (110)). However, the number of managed client settings for the policy-based method would be 25,410 (number of clients (231) × number of client settings (110)) because the policy-based method can support multiple security profiles in a single realm.

If a new security profile is added, the number of managed client settings for the settings-based method would be 16,940 (number of clients (154) × number of client settings (110)) because a new realm would need to be created for the new security profile. Conversely, the number of managed client settings would be 0 for the policy-based method.

If the supported security profile is modified, the number of managed client settings for the settings-based method would be 16,940 (number of clients (154) × number of client settings (110)) because every client would need to be reconfigured for the modified security profile. However, the number of managed client settings would be 0 for the policy-based method.

### 5.4 Discussion

According to the results of the evaluation of scenario 1, the policy-based method can apply the same security profile to authorization and token requests of the same authorization code flow.

According to the result of the evaluation of scenario 2, the policy-based method can apply different security profiles to a different request from a client based on its scope value, indicating the type of API that the client wanted to access. Thus, the policy-based method can support multiple security profiles in a single realm.

Therefore, it was concluded that the policy-based method considered the characteristics of an open banking use case and resolved the problems of increasing managerial costs for an authorization server administrator when the settings-based method is used for applying a security profile.

According to the results, the policy-based method decreases managerial costs in relation to the client settings-based method.

The policy-based method can be utilized not only in use cases like open baking but also in zero trust architectures (ZTAs) [36]. If an organization’s internal network adopts a ZTA and an access token in OAuth 2.0 is used by a client to access an organization’s resource, the process of issuing the access token needs to be secured by applying a security profile like FAPI because any communication is not trusted in the network. Therefore, the same problems in open banking may occur in the network that adopts ZTA if there are many clients and many security profiles need to be used in the network. The policy-based method can resolve the problems in ZTA.

## 6. Related Work

The proposed method needs to judge if a security profile is applied or not after receiving a client request. This can be considered access control. In access control, a server needs to judge if the request is allowed or not after receiving a request to access a resource. Therefore, access control methods were investigated to find the most appropriate one to apply to the proposed method.

ABAC is general model so that it can represent wide range of access control. ABAC was formalized in [37](2005). It was also proved that the formalized ABAC

model can represent Mandatory Access Control (MAC) and Discretionary Access Control (DAC) models in [37](2005). The minimum requirements of ABAC for representing MAC, DAC, and Role Based Access Control (RBAC) was derived and formalized in [38].

ABAC can represent access control based on an access token in OAuth 2.0 [1]. In OAuth 2.0, a client accesses a resource owner's resource with an access token showing the fact that the resource owner granted access to its resource to the client, so the client, the resource owner's resource, and the access token can be represented as a subject, an object, and an environment in ABAC respectively. When receiving the access request, the resource server holding the resource can judges whether the access request to the resource is accepted by itself or by querying an authorization server to determine the active state of the access token and to determine the meta-information about this token by OAuth 2.0 Token Introspection [39], so the resource server and the authorization server can be represented as PDP. Finally, the resource server returns a normal response or an error response to the client, so the resource server can be considered as a PEP.

ABAC was applied to systems using OAuth 2.0. ABAC was applied to OAuth 2.0 based authorization in distributed financial systems [40], where a client was represented as a subject in ABAC and a resource owner's resource was represented as an object in ABAC. ABAC was applied to OAuth 2.0 based Internet of Things (IoT) virtualization platform [41], where a user or service was represented as a subject in ABAC and a REST resource of the Master-Controller API was represented as an object in ABAC. OAuth 2.0 with ABAC was used for the collaborative sharing of equipment at construction sites [42], where a user device was represented as a subject in ABAC and a service was represented as an object in ABAC.

The proposed method applies ABAC because all entities that the proposed method interacts with can be represented as entities of ABAC. The proposed method needs to use contents of a client request to determine whether a security profile is applied or not. The contents of a client request can be represented as environment attributes in ABAC. A client can be represented as a subject in ABAC. Authorization and token endpoints that receive requests from a client can be represented as objects in ABAC and PEP. The proposed method can work as PDP.

The policies used in ABAC were categorized into logic-based and enumerated policies, as in [43]. While the policy machine described in [44] was mentioned as an enumerated policy, Label Based Access Control was newly proposed as a simple type of policy machine in [43]. An enumerated policy has an advantage against a logic-based policy regarding the amount of computation required to apply the policy. However, the capability of an enumerated policy to represent a policy is limited. Therefore, the proposed method applied a logic-based policy.

Hierarchical Group and ABAC introduces hierarchies of objects and subject groups to avoid setting the same attribute values to many objects or subjects and was proposed and formalized in [45] using Kleene K3 as logic to support a ternary value for its policy language. There was a case where a policy of the proposed method was not evaluated, so the proposed method applied a ternary value.

A logic-based policy problem regarding policy review was suggested in [43]. Policy review involves clarifying which attribute values return a positive result. Policy review in a logic-based policy using propositional logic is similar to the satisfiability problem in propositional logic, which is NP-complete. However, policy review in a logic-based policy using first-order logic is similar to the satisfiability problem in first-order logic, which is undecidable. Therefore, it is difficult to complete a policy review if a logic-based policy is used. This issue is recognized but not treated in this study.

The study of policy conflict is discussed in [46], [47] and defined as the situation where one policy accepts every request while another policy rejects all requests. Thus, policies do not work properly. Policy anomaly is defined in [47] as the situation where all policies accept or reject every request. Multiple policies are redundant if policy anomaly occurs. A method of constructing policies without policy conflict and anomaly was proposed in [47] by using the Fine-grained Integration Algebra (FIA) [48]. FIA is defined in [48] as an algebra for combining multiple trinary logic policies that demonstrate no policy conflicts or anomalies for policies of Extensible Access Control Markup Language (XACML) [49], which is an actual implementation of ABAC as an open standard specification. Conflict and anomaly of rules that comprise a policy are described in [50]. However, it does not describe how to find them generally. A policy conflict problem that occurs due to the nature of a logic-based policy was mentioned in [51]. A general idea of resolving policy conflict by using "metapolicies" was also suggested in [51]. The need for detecting and resolving policy conflict and anomaly is recognized but not treated in this study.

Some products of an authorization server can implement the policy-based method for multiple purposes. Keycloak 15 [6], ForgeRock AM 7.1 [7], Okta Classic Engine 2022.03 [8], and Auth0 [9] support the policy-based method for access control on resource servers' resources. Keycloak 15 and ForgeRock AM 7.1 use User Managed Access 2.0 [52], [53] but also have their own mechanisms for access control, which can work as a PDP. ForgeRock AM 7.1 [54] also supports XACML [49]. ForgeRock AM 7.1 [55] and Okta Classic Engine 2022.03 [8] support the policy-based method for access control on pages hosted by a client that a user accesses through a browser. Amazon Cognito supports the policy-based method for access control on AWS resources [56]. Keycloak 15 [57] and PingFederate 11 [58] support the policy-based method for registering and modifying a client using dynamic client registration [4]. PingFederate 11 [59] implemented the policy-based method for determining which type of authentication methods should be applied dynamically. However, they were not used for applying security profiles. Conversely, the policy-based method

was proposed for applying security profiles.

## 7. Conclusion

This study proposed a policy-based method for open banking security profiles to solve the problems of increasing managerial costs and the need to modify source code on an authorization server. When the settings-based method is used for applying a security profile when an authorization server is used in an open banking use case where many clients need to be managed and a different security profile needs to be applied to different types of APIs.

The proposed policy-based method was implemented to Keycloak with FAPI security profiles to confirm that these problems could be prevented. Evaluation scenarios were executed using the implementation that confirmed that the policy-based method resolved the problems.

This implementation of the policy-based method and security profiles (FAPI1-baseline, FAPI1-advanced and FAPI-CIBA) were contributed to Keycloak. Before the contribution, it was confirmed that the implementation passed the conformance tests of the OpenID Conformance Suite provided by the OIDF [60]. Subsequently, the implementation was added to the Keycloak upstream repository, reviewed by Keycloak maintainers, and merged into Keycloak's main branch in Keycloak [61]. Keycloak 15 or later includes this contribution, which is publicly available for use. Evidence of this contribution to Keycloak is described in Appendix.

Keycloak 15 was certified by OIDF as an OpenID Provider supporting FAPI 1.0 Advanced [62], FAPI-CIBA [63], Open Banking Brazil FAPI 1.0 [64], and Australia Consumer Data Right [65] security profiles by utilizing the contributed policy-based method, FAPI 1.0 Baseline, FAPI 1.0 Advanced, and FAPI-CIBA security profiles [66].

In the future, other security profiles, such as FAPI 2.0 [67], will be implemented using the policy-based method and contributed to Keycloak.

## Acknowledgments

We would like to express our gratitude to Stian Thorgersen, a project lead at Keycloak, Marek Posolda from the Keycloak development team, FAPI-SIG's members, and the Keycloak community. Their encouragement to have further discussion provided us with advice about our client policies implementation.

A part of this research was supported by JSPS Grants-in-Aid for Scientific Research (JP19H05579).

### References

[1] D. Hardt, "RFC 6749 The OAuth 2.0 authorization framework," IETF, https://datatracker.ietf.org/doc/html/rfc6749, Oct. 2012.
[2] T. Lodderstedt, M. McGloin, and P. Hunt, "RFC 6819 OAuth 2.0 threat model and security considerations," IETF, https://datatracker.ietf.org/doc/html/rfc6819, Jan. 2013.
[3] M. Jones, N. Sakimura, and J. Bradley, "RFC 8414 OAuth 2.0 authorization server metadata," IETF, https://datatracker.ietf.org/doc/html/rfc8414, June 2018.
[4] J. Richer, M. Jones, J. Bradley, M. Machulak, and P. Hunt, "RFC 7591 OAuth 2.0 dynamic client registration protocol," IETF, https://datatracker.ietf.org/doc/html/rfc7591, July 2015.
[5] F. Fernández, Á. Alonso, L. Marco, and J. Salvachúa, "A model to enable application-scoped access control as a service for IoT using OAuth 2.0," Proc. 2017 20th Conference on Innovations in Clouds, Internet and Networks (ICIN), pp.322–324, 2017.
[6] "Policy," Keycloak 15 Authorization Services Guide, https://www.keycloak.org/docs/15.0/authorization_services/index.html#policy, June 14. 2022.
[7] "How to manage UMA policies," ForgeRock AM 7.1 User-Managed Access (UMA) 2.0 Guide, https://backstage.forgerock.com/docs/am/7.1/uma-guide/managing-uma-policies.html, June 14. 2022.
[8] "What are policies," okta Developer, https://developer.okta.com/docs/concepts/policies/, June 14. 2022.
[9] "Authorization policies," auth0 docs, https://auth0.com/docs/manage-users/access-control/authorization-policies, June 14. 2022.
[10] "Keycloak," Keycloak.org, https://www.keycloak.org/, accessed June 14. 2022.
[11] Boleslaw Dawidowicz, "Keycloak proposal for submission to CNCF," Cloud Native Computing Foundation, https://github.com/cncf/toc/pull/405, accessed June 14. 2022.
[12] Y, Nakamura, and K. Enomoto, "Authentication and authorization based on OSS for secure system interoperation," https://www.hitachi.com/rev/archive/2020/r2020_05/05a04/index.html, accessed: 16/12/2021.
[13] N. Sakimura, J. Bradley, and E. Jay, "Financial-grade API security profile 1.0 - Part 1: Baseline," The OpenID Foundation, https://openid.net/specs/openid-financial-api-part-1-1_0-final.html, accessed March 20. 2022.
[14] N. Sakimura, J. Bradley, and E. Jay, "Financial-grade API security profile 1.0 - Part 2: Advanced," The OpenID Foundation, https://openid.net/specs/openid-financial-api-part-2-1_0.html, accessed March 20. 2022.
[15] D. Tonge, J. Heenan, T. Lodderstedt, and B. Campbell, "Financial-grade API: Client Initiated Backchannel authentication profile," The OpenID Foundation, https://openid.net/specs/openid-financial-api-ciba-ID1.html, accessed March 20. 2022.
[16] "Open banking security profiles," The Open Banking Implementation Entity, https://standards.openbanking.org.uk/security-profiles, accessed March 20. 2022.
[17] "Consumer data right security profile," Australian Competition and Consumer Commission, https://consumerdatastandardsaustralia.github.io/standards/#security-profile, accessed March 20. 2022.
[18] Open Banking Brasil GT Security, "Open Finance Brasil financial-grade API security profile 1.0 implementers draft 3," The Open Banking Brasil Initial Structure, https://openbanking-brasil.github.io/specs-seguranca/open-banking-brasil-financial-api-1_ID3.html, accessed March 20. 2022.
[19] T. Norimatsu, Y. Nakamura, and T. Yamauchi, "Flexible method for supporting OAuth 2.0 based security profiles in Keycloak," Open Identity Summit 2022 (OID 2022), Lect. Notes Informatics (LNI), vol.P-325, pp.87–98, 2022, DOI: 10.18420/OID2022_07.
[20] "Open banking read-write API profile - v3.1.10," The Open Banking Implementation Entity, https://openbankinguk.github.io/read-write-api-site3/v3.1.10/profiles/read-write-data-api-profile.html, accessed Aug. 2. 2022.
[21] "Regulated providers," The Open Banking Implementation Entity, https://www.openbanking.org.uk/regulated-providers/?query=directories&filter-search=&filter-provider-type=third-party-providers&filter-sort=0, accessed Aug. 2. 2022.
[22] "Quem participa - Instituicao transmissora / receptora de dados," The Open Banking Brasil, https://openbankingbrasil.org.br/quem-participa/?marca=&modalidade=DADOS, accessed Aug. 2. 2022.

[23] "Quem participa - Instituicao detentora de conta," The Open Banking Brasil, https://openbankingbrasil.org.br/quem-participa/?marca=&modalidade=CONTA, accessed Aug. 2. 2022.
[24] "Core concepts and terms," Keycloak 15 Server Administration Guide, https://www.keycloak.org/docs/15.0/server_admin/index.html#core-concepts-and-terms, accessed June 14. 2022.
[25] "Glossary," ForgeRock AM 7.1 Setup Guide, https://backstage.forgerock.com/docs/am/7.1/setup-guide/openam-glossary.html#openam-glossary, accessed June 14. 2022.
[26] "Multi-tenant solutions," okta Developer, https://developer.okta.com/docs/concepts/multi-tenancy/, accessed June 14. 2022.
[27] "Create tenants," auth0 docs, https://auth0.com/docs/get-started/auth0-overview/create-tenants, accessed June 14. 2022.
[28] "What is Amazon Cognito?," Amazon Cognito Developer Guide, https://docs.aws.amazon.com/cognito/latest/developerguide/what-is-amazon-cognito.html, accessed June 14. 2022.
[29] "OIDC Clients," Keycloak 15 Server Administration Guide, https://www.keycloak.org/docs/15.0/server_admin/index.html#_oidc_clients, accessed June 14. 2022.
[30] "Client registration," ForgeRock AM 7.1 OAuth 2.0 Guide, https://backstage.forgerock.com/docs/am/7.1/oauth2-guide/oauth2-register-client.html, June 14. 2022.
[31] "Configuring OAuth clients," PingFederate Server 11 Administrator's Reference Guide, https://docs.pingidentity.com/bundle/pingfederate-110/page/roj1564002966901.html, June 14. 2022.
[32] "Application settings," auth0 docs, https://auth0.com/docs/get-started/applications/application-settings, June 14. 2022.
[33] "Add OAuth 2.0 client application," okta Developer, https://developer.okta.com/docs/reference/api/apps/#add-oauth-2-0-client-application, June 14. 2022.
[34] N. Sakimura, J. Bradley, and N. Agarwal, "RFC 7636 Proof key for code exchange by OAuth public clients," IETF, https://datatracker.ietf.org/doc/html/rfc7636, Sept. 2015.
[35] N. Sakimura, J. Bradley, M.B. Jones, B. de Medeiros, and C. Mortimore, "OpenID Connect Core 1.0 incorporating errata set 1," The OpenID Foundation, https://openid.net/specs/openid-connect-core-1_0.html, accessed March 20. 2022.
[36] S. Rose, O. Borchert, S. Mitchell, and S. Connelly, "NIST special publication 800-207 zero trust architecture," National Institute of Standards and Technology, 2022.
[37] E. Yuan and J. Tong, "Attributed based access control (ABAC) for web services," Proc. IEEE International Conference on Web Services (ICWS)., vol.856, pp.561–569, 2005.
[38] X. Jin, R. Krishnan, and R. Sandhu, "A unified attribute-based access control model covering DAC, MAC and RBAC," DBSec'12: Proc. 26th Annual IFIP WG 11.3 conference on Data and Applications Security and Privacy, pp.41–55, 2012.
[39] J. Richer, "RFC 7662 OAuth 2.0 token introspection," IETF, https://datatracker.ietf.org/doc/html/rfc7662, Oct. 2015.
[40] A.S. Li, R. Safavi-Naini, and P.W.L. Fong, "A capability-based distributed authorization system to enforce context-aware permission sequences," SACMAT '22: Proc. 27th ACM on Symposium on Access Control Models and Technologies, New York, NY, USA, ACM, pp.195–206, June 2022, DOI: 10.1145/3532105.3535014
[41] P. Gonzalez-Gil, A.F. Skarmeta, and J.A. Martinez, "The security framework of Fed4IoT," CCIoT '20: Proc. Workshop on Cloud Continuum Services for Smart IoT Systems, pp.1–6, Nov. 2020, DOI: 10.1145/3417310.3431396
[42] U. Bodin, A. Christoffersson, A. Chiquito, J. Rodahl, and K. Synnes, "Application-scoped access control for the construction industry," Proc. 2021 26th IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp.1–8, 2021, DOI: 10.1109/ETFA45728.2021.9613645
[43] P. Biswas, R. Sandhu, and R. Krishnan, "Label-based access control: An ABAC model with enumerated authorization policy," Proc. 2016 ACM International Workshop on Attribute Based Access Control, pp.1–12, March 2016.
[44] D. Ferraiolo, V. Atluri, and S. Gavrila, "The Policy Machine: A novel architecture and framework for access control policy specification and enforcement," J. Syst. Archit., vol.57, no.4, pp.412–424, April 2011.
[45] D. Servos and S.L. Osborn, "HGABAC: Towards a formal model of hierarchical attribute-based access control," Foundations and Practice of Security, Springer, pp.187–204, 2014.
[46] E.C. Lupu and M. Sloman, "Conflicts in policy-based distributed systems management," IEEE Trans. Softw. Eng., vol.25, no.6, pp.852–869, Nov.-Dec. 1999.
[47] M. Yahiaoui, A. Zinedine, and M. Harti, "Deconflicting policies in attribute-based access control systems," 2018 IEEE 5th International Congress on Information Science and Technology (CiSt), 21-27 Oct. 2018.
[48] P. Rao, D. Lin, E. Bertino, N. Li, and J. Lobo, "An algebra for fine grained integration of XACML policies," SAC MAT 09: Proc. 14th ACM symposium on Access control models and technologies, 2 New York, NY, USA, ACM, pp.63–72, June 2009.
[49] E. Rissanen et al., "Xtensible Access Control Markup Language (XACML) Version 3.0 Plus Errata 01," OASIS Standard, 2012.
[50] K. Vijayalakshmi and V. Jayalakshmi, "Identifying considerable anomalies and conflicts in ABAC security policies," 2021 5th International Conference on Intelligent Computing and Control Systems (ICICCS), 6-8 May 2021.
[51] V.C. Hu, D. Ferraiolo, R. Kuhn, A. Schnitzer, K. Sandlin, R. Miller, and K. Scarfone, "Guide to attribute based access control (ABAC) definition and considerations," NIST Special Publication, 800:162, 2014.
[52] E. Maler, M. Machulak, and J. Richer, "User-Managed Access (UMA) 2.0 grant for OAuth 2.0 Authorization," Kantara Initiative, https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.html, accessed March 22. 2022.
[53] E. Maler, M. Machulak, and J. Richer, "Federated authorization for User-Managed Access (UMA) 2.0," Kantara Initiative, https://docs.kantarainitiative.org/uma/wg/rec-oauth-uma-grant-2.0.
html, accessed March 22. 2022.
[54] "Importing and exporting policies," ForgeRock AM 7.1 Authorization Guide, https://backstage.forgerock.com/docs/am/7.1/authorization-guide/import-export-policy.html, June 14. 2022.
[55] "About authorization and policy decisions," ForgeRock AM 7.1 Authorization Guide, https://backstage.forgerock.com/docs/am/7.1/authorization-guide/what-is-authz-decision.html, June 14. 2022.
[56] "Using attributes for access control as a form of attribute-based access control," Amazon Cognito Developer Guide, https://docs.aws.amazon.com/cognito/latest/developerguide/attributes-for-access-control.html, June 14. 2022.
[57] "5.8. Client registration policies," Keycloak 15 Securing Applications and Services Guide, https://www.keycloak.org/docs/15.0/securing_apps/index.html#_client_registration_policies, June 14. 2022.
[58] "Configuring a response type constraints instance," PingFederate Server 11 Administrator's Reference Guide, https://docs.pingidentity.com/bundle/pingfederate-110/page/msr1564002992437.html, June 14. 2022.
[59] "Authentication policies," PingFederate Server 11 Administrator's Reference Guide, https://docs.pingidentity.com/bundle/pingfederate-110/page/jul1564002986701.html, June 14. 2022.
[60] "OpenID conformance suite," OpenID Foundation, https://openid.net/certification/about-conformance-suite, accessed June 14. 2022.
[61] "Client policies and financial-grade API (FAPI) Support," Keycloak.org, https://www.keycloak.org/docs/latest/release_notes/index.html#client-policies-and-financial-grade-api-fapi-support, accessed June 14. 2022.
[62] "Certified Financial-grade API (FAPI) OpenID Providers - Financial-grade API (FAPI) 1.0 Final - FAPI 1 Advanced Final (Generic)," OpenID Foundation, https://openid.net/certification/#FAPI_OPs, accessed June 14. 2022.

[63] "Certified Financial-grade API Client Initiated Backchannel Authentication Profile (FAPI-CIBA) OpenID Providers," OpenID Foundation, https://openid.net/certification/#FAPI-CIBA_OPs, accessed June 14. 2022.
[64] "Certified Financial-grade API (FAPI) OpenID Providers - Financial-grade API (FAPI) 1.0 Final - Brazil Open Banking (Based on FAPI 1 Advanced Final)," OpenID Foundation, https://openid.net/certification/#FAPI_OPs, accessed June 14. 2022.
[65] "Certified Financial-grade API (FAPI) OpenID Providers - Financial-grade API (FAPI) 1.0 Final - Australia CDR (Based on FAPI 1 Advanced Final)," OpenID Foundation, https://openid.net/certification/#FAPI_OPs, accessed June 14. 2022.
[66] M. Posolda, "Keycloak certified as FAPI and Brazil Open Banking provider," Keycloak.org, https://www.keycloak.org/2022/01/fapi, accessed June 14. 2022.
[67] D. Fett, "FAPI 2.0: A high-security profile for OAuth and OpenID connect," Proc. Open Identity Summit 2021, pp.71–81, 2021.

## Appendix: Evidence of our Contribution to Keycloak

The following list includes the pull-requests sent by us for the policy-based method and security profile implementation and merger to Keycloak.

https://github.com/keycloak/keycloak/pull/7104, 7232, 7278, 7364, 7391, 7395, 7410, 7419, 7423, 7501, 7503, 7592, 7594, 7598, 7601, 7615, 7628, 7629, 7659, 7664, 7679, 7680, 7682, 7698, 7699, 7723, 7727, 7732, 7735, 7780, 7907, 7910, 7914, 7915, 7953, 8002, 8003, 8004, 8005, 8006, 8010, 8016, 8017, 8028, 8070, 8083, 8103, 8144, 8223, 8237, 8238, 8239, 8240, 8241, 8242, 8252, 8277, 8278, 8283, 8284, 8285, 8293, 8294, 8295, 8302, 8308, 8309, 8597, 9490, 9526, 9527

The following list includes the acknowledgement of our contribution to Keycloak's release notes from the Keycloak community:

- Keycloak version 12: FAPI RW support and initial support to Client policies
  https://www.keycloak.org/docs/latest/release_notes/index.html#fapi-rw-support-and-initial-support-to-client-policies
- Keycloak version 13: OpenID Connect Client Initiated Backchannel Authentication (CIBA)
  https://www.keycloak.org/docs/latest/release_notes/index.html#openid-connect-client-initiated-backchannel-authentication-ciba
- Keycloak version 14: Client Policies and Financial-grade API (FAPI) Support
  https://www.keycloak.org/docs/latest/release_notes/index.html#client-policies-and-financial-grade-api-fapi-support
- Keycloak version 15: Financial-grade API (FAPI) Improvements, FAPI CIBA and Open Banking Brasil
  https://www.keycloak.org/docs/latest/release_notes/index.html#financial-grade-api-fapi-improvements-fapi-ciba-and-open-banking-brasil

**Takashi Norimatsu** received his B.S. degree in science from National Institution for Academic Degrees and Quality Enhancement of Higher Education in 2000, and M.S. degree in mathematical engineering from University of Tsukuba in 2001. He has been working for Hitachi, Ltd. since 2001, and is also studying at Okayama University to obtain a Ph.D. degree. He is a member of IPSJ.

**Yuichi Nakamura** received his B.S. and M.S. degrees in physics from University of Tokyo in 1999 and 2001, M.S. degree in computer science from The George Washington University in 2006, and Ph.D. degree in computer science from Okayama University in 2016. He worked for Hitachi Solutions over 2001–2015 and has been working for Hitachi, Ltd. since 2016. He is a member of IPSJ.

**Toshihiro Yamauchi** received B.E., M.E. and Ph.D. degrees in computer science from Kyushu University, Japan in 1998, 2000 and 2002, respectively. In 2001, he became a Research Fellow of the Japan Society for the Promotion of Science. In 2002, he became a Research Associate in Faculty of Information Science and Electrical Engineering at Kyushu University. He has served as associate professor of Graduate School of Natural Science and Technology at Okayama University since 2005, and has been serving as professor of Graduate School of Natural Science and Technology at Okayama University since 2021. His research interests include operating systems and computer security. He is a member of IPSJ, IEICE, ACM, USENIX, and IEEE.