AU2EU

Architecture for Integrating Authorization with eAuthentication

Dhouha Ayed (THA), Michael Hitchens (MQ), Dieter Sommer (IBM) Daniel Kovacs (IBM) Zubair Baig (ECU)

This project has received funding from the European Union's Seventh Framework Programme for research, technological development and demonstration under grant agreement no: 611659

Project Information

Authentication and Authorization for Entrusted Unions

Project number:	611659
Strategic objective:	ICT-2013.1.4.e
Starting date:	2013-12-01
Ending data:	2015-11-30
Website:	http://au2eu.eu/

Document Information

Title:	Architecture for Integrating Authorization with eAuthentication				
ID:	D3.2.1	Type:	R	Dissemination level:	PU
		Month:	M12	Release date:	30.November.2014

Deliverable Description

Specifies administration mechanisms for unifying authorization based on ontology: Specifies a translation mechanism from authorization policy to the plan for authentication of claims

Contributors, Editor & Reviewer Information

Contributors (person(partner): sections)	Dhouha Ayed(THA) entire document except 6.5 Michael Hitchens(MQ) entire document except 6.5 Dieter Sommer(IBM) entire document except 6.5 Daniel Kovacs (IBM) 6.4 Zubair Baig (ECU) 6.5
Editor (person/partner)	Michael Hitchens/MQ
Reviewer (person/partner)	Saeed Sedghi/PRE

Release History

Release number	Date issued	Milestone*	SVN version	Release description / changes made
1	18th August 2014	PCOS		Outline and description of sections and subsections
2	21st November 2014	Proposed		Full document for internal review, with full content
3	30th November 2014	External		Full description, with changes after internal review

* The project uses a multi-stage internal review and release process, with defined milestones. Milestone names include abbreviations/terms as follows:

1. PCOS = ”Planned Content and Structure” (describes planned contents of different sections)
2. Intermediate: Document is approximately 50% complete – review checkpoint
3. External For release to commission and reviewers;
4. proposed: Document authors submit for internal review
5. revised: Document authors produce new version in response to internal reviewer comments
6. approved: Internal project reviewers accept the document
7. released: Project Technical Manager/Coordinator release to Commission Services

AU2EU Consortium

Full Name	Abbreviated Name	Country
Technische Universiteit Eindhoven	TU/e	Netherlands
Philips Electronics Nederland B.V.	PRE	Netherlands
Bicore Services B.V.	BIC	Netherlands
NEC Europe LTD	NEC	United Kingdom
IBM Research GMBH	IBM	Switzerland
Deutsches Rotes Kreuz	DRK	Germany
Thales Communications & Security SAS	THA	France
Commonwealth Scientific and Industrial Research Organisation	CSO	Australia
Edith Cowan University	ECU	Australia
Royal Melbourne Institute of Technology	RMI	Australia
University of New South Wales	NSW	Australia
Macquarie University	MQ	Australia

Table 1: Consortium Members

Table of Contents

List of Figures

1 Executive Summary

This deliverable presents the architectural framework that enables the integration of the authorization chain with authentication mechanisms. The architecture allows users to access the offerings of service providers by releasing the minimum necessary amount of their personal information, in such a way that user privacy is preserved as much as possible.

The architecture of this deliverable enables users to obtain policies relevant to the requested access from the service provider and then present claims that may permit the access. Service providers can select which policy or polices should be provided to users. The policies presented to the user may allow only one means of fulfillment (for example, access is allowed based on the user presenting some evidence related to their age) or they may provide the user with some choice (for example, access is allowed based on the user presenting some evidence related to either their age or their address of residence). The user will then provide a claim against the policy provided. If polices presented allow the user a choice, they will formulate and present a claim against the option they prefer. The claim is then returned to the service provider, which assesses whether the presented claim satisfies the requirements for access to the service. This approach allows a release of information from both parties that is both minimal and under their control.

User credentials, and the creation of claims, are managed by an authentication system. The policies governing service provider resources are managed through an authorization system. For development purposes the project uses ABC4Trust and XACML, respectively. Regardless of the actual technology employed, the syntax used for policies, and even what policies can be expressed, will likely differ between the authentication and authorization mechanisms. Hence the need for translation between, and integration of, the two mechanisms. The proposed architecture sites this work on the service provider. This is in the interest of providing a thin-client solution. The design is driven by the principles of minimal changes to existing implementations (in the interests of simplicity and assisting adoption) and in preserving the service provider authorization system as the ultimate controller of access.

This deliverable identifies design questions that need to be answered in creating an architecture that integrates disparate authentication and authorization systems. A number of possibilities are presented and analysed. While they have drawbacks compared to the proposed architecture, the analysis is useful in illuminating the issues involved.

The deliverable also presents a summary of the potential exploits that may be run against the proposed architecture. The actual resilience against malicious attacks will be the subject of a later deliverable.

2 About this Document

2.1 Role of the deliverable

The objective of this task is to integrate the authorization chain with the eAuthentication framework, possibly for independently-administrated security domains. The exchange methods and architecture for the translation of the authorization policies into the corresponding required authentication attributes and the production of corresponding claims/evidence will be defined. These authentication claims will be used to determine the access rights of the user to the requested services/resources.

2.2 Relationship to other AU2EU deliverables

This document takes into account the requirements defined in D1.2.1 “Requirements specification document” as well as the scenarios specified in D1.1.1 “Detailed descriptions of use cases”. D1.4.1 “Reference architecture for eAuthentication & eAuthorization” provides recommendations for the overall architecture of AU2EU, to which the authorization platform must conform. This deliverable specifies an integrated architecture solution of eAuthentication and eAuthorization. This architecture will be used as background to implement the authorization platform that will be delivered in D3.3.1 and D3.3.2.

2.3 Relationship to other versions of this deliverable

This version of the document has been amended based on internal review and supersedes all previous versions.

2.4 Structure of this document

This document discusses the background, design issues and design proposals for the integration of the authentication and authorization mechanisms. Section 3 gives relevant background and context, including the underlying technology. Section 4 identifies and discusses various issues that need to be considered in the design of the architecture. Section 5 discusses some design proposals, in terms of the identified design issues, and their advantages and disadvantages. Section 6 describes the proposed architecture and approach. Section 7 concludes the document.

3 Context and Challenges

Authentication and authorization are fundamental concepts in the protection of resources. Traditionally these two concepts, and the mechanisms that support them, are treated as separate entities, with the only relation between them being that (generally) a user needs to be authenticated before she can be authorized to access a resource.

Such a separation results in:

1. users needing to fully identify themselves before access can occur, which may lead to violation of least privilege [1] and user privacy [2].

2. the information handled by the two mechanisms also being separate.

The violations of least privilege and user privacy flow from the strongly separated nature of traditional authentication and authorization mechanisms. In such systems authentication must occur before any access control requests are made, the authentication process must require all information that may be relevant to any future request. As the user must then, perforce, reveal extensive information it is easy to see how this does not confirm to least privilege and has severe implications for user privacy. The developers of authorization mechanisms have little incentive to provide fine-grained policy approaches, as the default assumption is that extensive information about the user will be available following authentication.

Both concepts, i.e., least privilege and user privacy, require that the user reveal only the minimal amount of personal information which is necessary for a given access decision to be made. This minimal approach ensures that the deciding entity receives only the information it requires and no more and that users release only a portion of their personal information so that their privacy is not violated. The information required for each access needs to be decided on a per-request basis, as the information necessary to assess one request may bear little relation to the information relevant to another request. In traditional authentication/authorization based systems the user must reveal all information up front, on authentication, in such a way that a successful identification can then be applied to any future request for authorization. The user then has to reveal nothing further, as everything is already known to the authorization system. An analogous situation can be seen in systems which use any form of credential in an unmoderated fashion. That is, where the user has a set of credentials and must provide them in full when attempting any operation within the system. Access to a resource in all such systems requires the user to reveal all information that could possibly be relevant to the decision, not a minimally sufficient subset.

On the other hand, systems which enforce a strict separation between authentication and authorization often limit the flow of information between the two mechanisms. For example, information used in proving identity may not be used in assessing and enforcing authorization, as all that is provided to the latter mechanism is an assurance of identity, not the information that is used to arrive at that assurance.

User privacy and least privilege both require a minimal set of information to be provided. Given this minimal position it appears reasonable to assume that both authentication and authorization systems may need access to this information to make decisions based on their respective policies. This is especially true if authentication is not viewed as a one-time act, but is associated with individual accesses to resources. It therefore appears reasonable to attempt to integrate authentication and authorization mechanisms. This will allow the same information to be used by both mechanisms.

The concept of minimal release of information can be illustrated by the following example. Imagine a resource which has an associated policy that specifies access is available to those users that are over the age of 18 and resident in Europe. Information about a user’s residency and age will often be held in a credential which explicitly specifies the two values. For example, "Address:15 Boundary Rd, Tottenham, London, UK, Age: 36 years". It would not be unusual for such a credential to also include the user’s name (for example, “Name: Katrina Hastings”). Such a specific credential, containing rich and detailed information, is useful in satisfying a vast range of policies. However, in the case given in this example, the user credential reveals information about the user to the service provider that is not required to decide access permission to the resource, thus violating the user's privacy. Note that this can include details of the information (such as actual age) or information not even mentioned in the policy (in the example given, the user’s name). It would also likely require the service provider to expend processing time to locate the required information in the credential. The architecture proposed in this deliverable addresses these drawbacks, in such a way that credentials for the user are created on demand, including exactly the information (and no more) that is required to satisfy the policy. In the case given above the credential would simply attest to the user being a resident of Europe and over the age of 18. While it might be argued that the same effect could be achieved if the user had a range of credentials ready to provide to service providers, this approach has a number of drawbacks. First, the user would have to decide *in advance* which credentials to obtain (and be able to obtain them). Without knowledge of the range of policies in effect at service providers, this would be difficult to achieve. Second, even assuming that a range of credentials was obtained, storing and managing them would be problematic. By creating what are effectively disposable, single-use, credentials, the current proposal avoids these issues.

The architecture presented in this document integrates the authentication and authorization mechanism. In order to be standard compliant the chosen technology for authorization is XACML [3]. In the interest of being able to produce a working prototype in the minimum time, and also to allow for widespread adoption/implementation, the decision was taken to leverage off existing components wherever possible. For authentication the chosen technology is ABC4Trust [4]. ABC4Trust defines concrete components of a privacy-preserving identity management system and relies on attribute-based authentication claims. As such, the attributed-based XACML authorization framework, which allows making and enforcing the authorization decisions based on attributes provided by different Claims Providers, perfectly complements the ABC4Trust framework.

ABC4Trust is an architecture[1] for authentications systems that allows users to authenticate while preserving their privacy. That is, it adheres to the concept of user privacy. It is a credential based system. Credentials are provided to users by one or more *Issuers*. These credentials attest to certain information with respect to the user for whom the credential is issued. Access to resources and services are protected by *Verifiers*. These restrictions are expressed in terms of *presentation policies,* which define the information that must be contained in the credentials the users must present is access is to be obtained. An overview of the ABC4 trust interactions can be seen in figure 1. After obtaining a presentation policy from the verifier a user can derive *presentation tokens* from their issued creden-

[1] The ABC4Trust project also provided a reference implementation of all components

tials that reveal appropriate information to the verifier. For more information on ABC4Trust the reader is referred to [4].

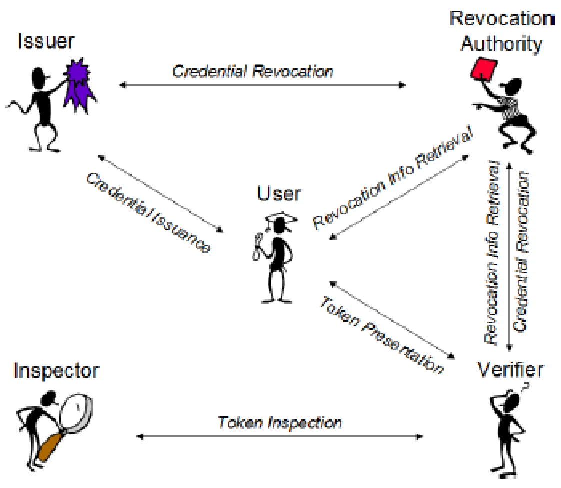

Figure 1: ABC4Trust Entities and Interactions diagram from [4]

The choice of XACML and ABC4Trust as the major component is not without drawbacks, as they are not designed to be compatible. Their policies are not expressed in the same language or format. The two technologies place different limits on the attributes used in evaluating policies. For example, both allow predicates within policies (e.g., that the users age must be greater than or equal to 18). But standard XACML expects a discrete value for the user's age (say, 22), whereas ABC4Trust allows the attribute itself to be a predicate (age >= 21). These differences have obvious implications for the complexity of the resulting integrated architecture.

The alternatives were either to extend XACML to match the expressive capacities of the ABC4Trust policy language or reconsider the use of either or both of these components and construct the system (either in whole or major part) from the ground up. Either of these approaches would have benefits in terms of design simplicity as the issues flowing from the lack of compatibility between XACML and ABC4trust would be avoided. However, any such new proposal, in an area where there is existing technology (and, in the case of XACML at least, one with quite widespread acceptance) would potentially face considerable difficulties in gaining widespread adoption.

Further discussion of the issues briefly covered in this section can be found in [5].

4 Design Issues in Integrating Authentication and Authorization

Any system that grants users access to resources has simple, well-known, high level structures. The user makes a request, potentially providing relevant information. The security systems of the service provider (or a third party) decide whether or not to grant the access requested, based on the resource and action requested, information provided with the request and any other relevant information available.

Given the decisions taken on the choice of technologies, as discussed in the previous section, users will employ the ABC4Trust system to manage their credentials and generate claims to accompany requests for access. The construction of these claims will be governed by the protection of user privacy, as that was a primary aim in the design of ABC4Trust. The service provider will employ XACML as the core of the processing of such requests. The high-level architecture can be seen in figure 2.

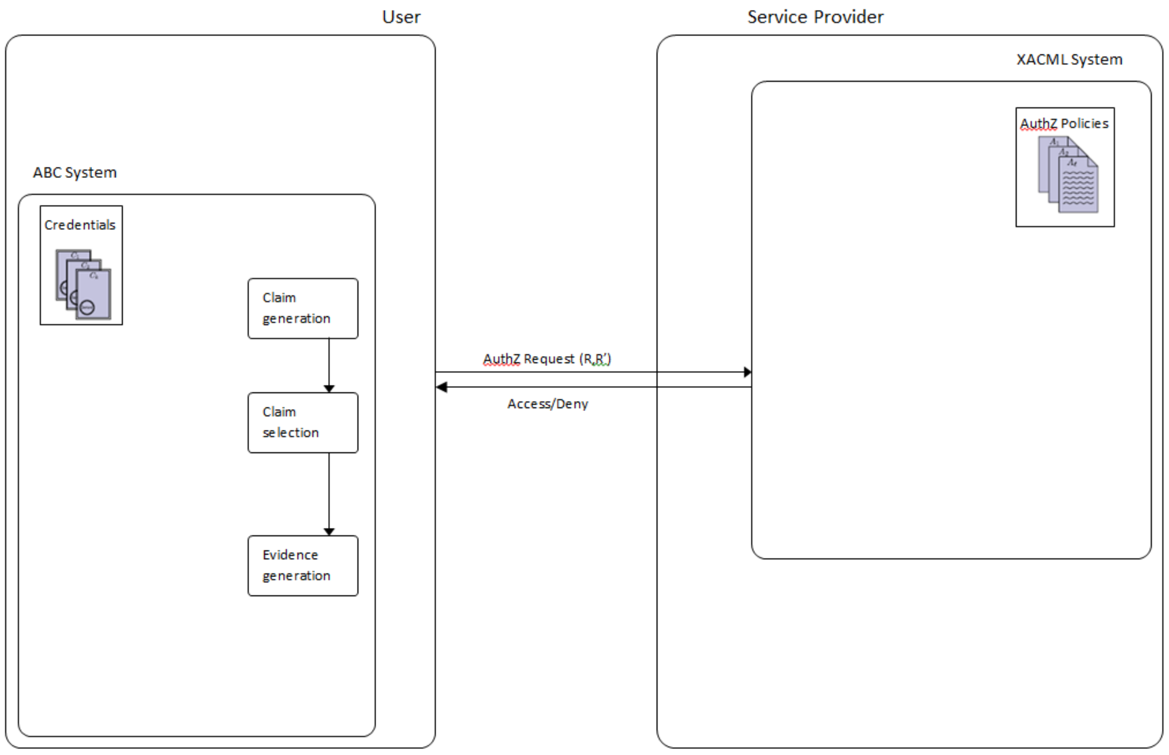

Figure 2: High Level Architecture

The high-level architecture leaves many questions unanswered. For example, the user request in figure 2 is labelled as an authorization (AuthZ) request as this is what the service provider's XACML system will require, yet the user only has ABC4Trust, which is essentially an authentication (AuthN) mechanism. Arriving at an architecture that will enable the integration of authentication and authorization mechanisms which have not been constructed for compatibility requires the consideration of a number of detailed design issues. Note that these design issues are independent of the actual implementation of the underlying mechanisms, but are heavily dependent on the design of those mechanisms.

In this section we discuss a number of design issues which need to be addressed in arriving at an architecture that will both obey the principles of least privilege and user privacy and enable the integration of the chosen underlying technologies. These decisions involve the high-level structure and content of messages between the user and service provider and the components within the user and service provider responsible for various functions. Note that these decisions, even those involving particular components, are addressed in terms of the high-level architectures of the chosen technologies and we consider them design decisions as they are independent of actual implementations of those technologies. Examination of these issues guided the design of the architecture presented here and which is described in section 6.

The first question to be considered is how the user makes an AuthZ request when their system only contains an AuthN system. This question could be avoided if the user system also included an AuthN system (such as XACML). However, this would increase processing and storage load on the user system. The questions of translation and integration would not be avoided, merely their location changed. We consider that the processing load in terms of making these decisions is better placed on the service provider, to enable thinner client software and retain service provider control over access decisions.

For ABC4Trust to generate relevant claims/evidence, an authentication policy is required that is expressed in a format that ABC4Trust can understand (i.e., a presentation policy). This policy needs to be obtained from the service provider. The presentation policy will refer to a specific service (as requested by the user). The user can only provide relevant claims once the presentation policy is revealed. The presentation policy cannot be provided unless the user specifies the requested service/resource. It is then clear that the interaction between user and service provider will not be the simple two-message exchange shown in figure 2. The user will have to make two requests, one where the requested resource/service is specified and another where the appropriate presentation token is supplied. However, either ordering of these requests is possible. That is, the user could make the initial request, identifying the target service/resource as either an AuthN request or an AuthZ request. An AuthZ request must be made, either directly from the user or by translating an AuthN request, so that it can be processed by XACML. The final request from the user (either AuthN or AuthZ) must be accompanied by the claims/evidence relevant to the requested access.

This leads to the first design decision:

1 Is the authorization (AuthZ) request made by the user before or after the authentication (AuthN) policy is obtained from the service provider?

Note that either answer implies, as noted above, that the actual exchange between the users and service provider requires four messages, not the two shown in figure 2. One pair of messages is as shown in figure 2, the authorization request, the other pair results in the user obtaining the relevant authentication policy. An authentication policy is required for use by the user's ABC4Trust system. The choice is essentially whether the communication between user and service provider implies a push of the AuthN policy by the service provider to the user in response to the AuthZ request or a pull of the AuthN policy by the user from the service provider prior to the AuthZ request. In either case the user does not see the actual AuthZ policy employed by XACML.

If the AuthZ request is made first, then the exchange can begin with a standard XACML request (or at least something similar). The initial request cannot be accompanied by appropriate claims, as the user is unaware of the relevant policy. As a result the initial request will be denied because the service provider does not have any credential from the user in this phase. The only exception to this would be where the requested access required no user information, i.e., it is for a public operation. The message flow would, in outline, be:

1: U->SP: AuthZ request (R,R’)

2: SP->U: Deny, AuthN policy

3: U->SP: AuthZ request (R,R’), Claim

4: SP-> U: Permit/Deny

U: user
SP: Service provider
R: resource
R’: action

The first response from the service provider denies access, as described above. The second response either permits or denies the access, depending on whether the provided claim fulfils the AuthZ policy.

If the initial request was framed as a request for the relevant AuthN policy then this would not be a standard XACML request. XACML requests are for actual access, not policies. One advantage of this approach is that users could request the relevant policy before deciding to request access. If the access required information that the user was unwilling to reveal then the actual access request would not be made. This can be achieved in the AuthZ first approach, by the user breaking off the communication between steps 2 and 3 (which means both approaches allow the user to obtain the policy without proceeding to a genuine request). It might also appear that the user could cache policies until access was required. However, ABC4Trust has freshness mechanisms built into its message structure that make this infeasible. This means that AuthN policy cannot be stored on the user’s side for any length of time before being used in connection with an AuthZ request to the server.

If the initial request is for the AuthN policy then the exchange would be as follows:

1: U->SP: AuthN policy request (R, R’)

2: SP->U: AuthN policy

3: U->SP: AuthZ request (R, R’), Claim

4: SP-> U: Permit/Deny

While the differences might be regarded as minor, two considerations lead us to favour the former option (initial request is an AuthZ request), as stated below:

Firstly, the AuthN policy request first approach requires two distinct interfaces to be maintained, one for whatever system is handling the AuthN policy provision, the other the XACML system. Secondly, the AuthZ request first approach allows all messages to be couched as (modified) XACML message exchanges. Note that messages 1/3 and 2/4 in the first sequence above are slight variations on each other (2 and 3 carrying extra information compared to their counterparts). This allows the same mechanisms to be used to support all messages.

Another approach is to couch all messages form the user as AuthN requests. Option 2 still requires distinct interfaces (one to obtain the policy, one to make the actual request) and presents no corresponding advantages. The requests being made are for access and will be controlled by and sent to the service provider’s XACML implementation. As will be seen later their processing will involve

ABC4Trust components running on the service provider. However the point of contact for the user on the service provider is the XACML system (specifically, the PEP) and therefore we consider an XACML (AuthZ) message as the preferred choice.

2 What component is responsible for supplying the AuthN policy to the User?

Regardless of the answer to design question 1, above, the user's authentication system requires an AuthN policy to be supplied at some point in the interaction. The user's ABC4Trust system will select evidence and generate a claim in response to an AuthN policy. This AuthN policy must be supplied to the user. Some component or components in the service provider's system must be responsible for providing the policy for inclusion in the initial response to the user. This might be a component completely outside the XACML system, or a (modified) component of the XACML system. The AuthN policy has to correspond to the XACML AuthZ policy used to control access to the resource. It is, in effect, a translation of that XACML policy for use by the authentication system (in our case ABC4Trust components).

If the component supplying the AuthN policy is outside the XACML system it might be a separate (relatively stand-alone) component or it might be some form of gateway wrapped around the XACML system. The former (standalone component) approach would be a reasonable if the AuthN request initiates the exchange. It makes less sense if the exchange begins with the AuthZ request. An initial AuthZ request will most likely be in XACML format, and such a request would be best handled by an XACML component or a gateway strongly integrated with XACML. If an XACML component is chosen then the PEP appears most suitable, as it handles communication with the user. This would potentially localize any changes necessary to an XACML implementation to the interface component. Other options, such as the PAP, PDP or the obligation service would require more significant alterations. Either the alterations would be to a number of components, as every component between the one selected and the user (including the PEP) would need to be altered to handle changes to messages between XACML components. Or the selected component would need to be altered to handle a separate exchange with the user. Opening up a component in this way appears undesirable as it adds an extra interface that needs to be designed, implemented and maintained.

It may be assumed that the gateway approach would allow the initial request to be intercepted before it reaches XACML and the appropriate policy provided to the user without involving the service provider's XACML system. However, if the request was for a public action (or could otherwise be decided upon by the PDP without a claim from the user) this should not occur. There is also the question of how the gateway obtains the AuthZ policy so that it can be translated to an AuthN policy. This implies that all requests should initially go to the PDP (i.e., on receipt of the first message in the first proposed sequence under issue 1). The gateway approach then defaults to how the translation is implemented, either by a wrapper or by a more integrated redesign. A standalone component does not fit with our choice to begin the exchange with an AuthZ request, for the reasons just discussed.

As all requests must go to the PDP it is clear that the XACML system on the service provider must be involved in all requests. This is sensible, as deciding on user requests is the function of XACML and voiding this responsibility begs the question of why use XACML at all. Given this it is possible to retain XACML as having primary responsibility for the user exchange and provision of the AuthN policy. This can be done through the PEP, which can make use of the additional components as necessary. This retains the overall function and design of XACML with the corresponding responsibilities. The details of this are discussed in section 6.

3 Do the message pairs between the user and Service Provider use the same structure or different ones?

As discussed under issue 1, there will need to be two pairs of exchanges between the user and the service provider in most cases. As also discussed briefly under that issue, it is possible to use the same structure for both the message pairs (or, conversely, to have different structures). Having the same structure would simplify the implementation. It also allows for the unusual case where a second pair is not required. This occurs when access can be allowed without provision by the user of any claims/credentials (in the form of an ABC4Trust presentation token). It appears attractive to have at least one of the requests from the user in the form of an AuthZ request (as that is what the system is implementing). This implies that, if the message pairs are to have the same structure, that structure must have the messages from the user be AuthZ requests. The simplest implementation would be if that structure was the (possibly modified) XACML request structure. It can be seen that the resolution of this issue is not independent of the other issues, as employing the XACML request/response structure as the basic message structure between User and Service Provider has implications for, at least, the question of the order of the AuthZ request and provision of the AuthN policy. Basing the exchange around a well-known mechanism (XACML) will simplify both maintenance and, hopefully, adoption of the system. If the message pairs did not share a structure then the individual messages could be tailored to their particular purpose. Note, for example, the drawback in the first sequence under issue one of the second message almost always being ‘deny’. Having different structures would require increased implementation/maintenance costs, as always accompanies increased variation. It is also only necessary if the AuthZ request is not the opening of the exchange.

It can be seen that resolution of this issue is dependent on other issues, particularly issue 1.

4 What part of the system verifies that the Authentication Policy (AuthN) is satisfied?

When the user supplies the Service Provider with claim/evidence generated by her authentication system in response to the AuthN policy supplied by the Service Provider there needs to be a check of whether that claim/evidence satisfies the policy. This check could be carried out wholly within the XACML system, wholly outside of it, or divided between the two. The policy and claims/evidence are ABC4Trust artifacts. If some or all of the policy is checked outside of the Service Provider’s XACML system it appears reasonable that an ABC4Trust implementation on the Service Provider do that checking.

Within the XACML system various components could be used for policy verification (e.g., PEP, PDP, Obligation service). Choosing the PIP fits, to some extent, with the XACML design, as it allows the PIP to extract information from the token (i.e., subject attributes) and send that information to the PDP for use in its own decision-making process. However, unless the nature of the information is limited (to what could be derived from the presentation token without inspection), or the PDP is heavily modified, the PIP would be required to be an ABC4Trust inspector [6]. This would compromise user privacy. The PIP is also not generally tasked with making decisions on policy, so this does not appear an attractive choice.

Selecting another XACML component, such as the PEP or Obligation service would require fewer changes to the internal communication of XACML, as the central communication (PEP/PIP/PDP) would not need to be ABC4Trust aware. However, these components again do not normally make decisions about policy and splitting this responsibility within XACML appears to needlessly complicate the XACML architecture. They would also need significant modification and possibly the use of inspection, which is undesirable as noted above. It would also likely mean that the PDP makes its decision without access to in-formation from the presentation token.

The PDP is an obvious candidate, as its role is policy (although in the standard case XACML policy) checking. One issue is that, if the PDP is used for checking the entire policy in detail, it would need to understand ABC4Trust policies and the necessary information (policy and claim) would need to be passed around between XACML components. This would require significant changes to a standard XACML implementation, but does have the advantage of having a single component that is responsible for policy checking.

The problem here is that the standard XACML cannot handle all attributes that can be expressed in ABC4Trust. This means that a standard PDP implementation cannot fully evaluate the complete range of claims that can be generated by ABC4Trust. There are also examples, such as using inspection, that are outside the range of standard XACML. This leaves two high-level options, which both merit further discussion:

1. Design the system so that the PDP can evaluate what is expressed in all the AuthN claims and polices that will be used.
2. Accept that the PDP cannot make a complete decision by itself.

If option one is taken only two possibilities appear to present themselves. First, restrict the AuthN policies used so that they only express policies that can be evaluated by a standard XACML PDP. The second choice is to extend the PDP (and potentially other XACML components and SAML) so that the expressive capabilities of ABC4Trust can be handled. While both have some superficial advantages, they both also have significant weaknesses. The first option (limiting the AuthN policies) ignores much of the privacy functionality offered by ABC4Trust. Predicates as attributes (such as “age greater than 18”) could not be used; the actual values (in this case the user’s age) would have to be revealed to the service provider. Such an answer begs the question of why ABC4Trust would be used at all, as a simpler system on the user side, fully compatible with XACML, would simplify many of the design and implementation problems. The second option (considerably altering an XACML implementation) raises significant implementation and acceptance issues and does not appear feasible within the time and resources available. Even if such an implementation could be realized within the scope of the project, it appears questionable whether widespread acceptance of such a modified XACML implementation could be achieved.

This leaves us with option 2 above. Like option 1 there are a number of ways to realize it. One is to take all responsibility away from the PDP for making this decision and giving it to another component. This could an ABC4Trust system on the user side or some altered component of XACML. Both limit and localize the changes needed to XACML. The first requires very few changes. The second somewhat more, but, depending on the component chosen, less than if the PDP itself is to be altered. The significant drawback is that by removing this responsibility from the PDP entirely the questions arises (similar to one in the preceding paragraph) of why we would need XACML at all if its primary function (policy evaluation) is not to be employed.

Another way to realize option 2 is to split the responsibility for evaluating whether the claim/evidence supplied by the user satisfies the policy. The PDP would evaluate whatever it was capable of evaluating, according to standard XACML; some other component would evaluate the ABC4Trust specific elements. This other component may be part of an ABC4Trust system or some (altered) XACML component. The first (by dint of being simplest) question is where does ultimate authority lie? Less technically, what makes the final decision? We would contend that the PDP is the preferred option here, as that is the component with that role in XACML. Taking that responsibility away again raises questions of why XACML is being employed. The second question is how does this other (as yet notional) component convey its decision to the PDP? While not trivial, we consider that this is more of an implementation decision and less of a design issue and can be ignored for purposes of this section. Finally there is the question of whether such a split system would be reliable.

As a conclusion, we do not contend that this question has an easy or clean answer. We do, however, contend that the issue addressed in this section has three real answers.

1. Alter the PDP and other parts of XACML to handle the full functionality of ABC4Trust policies and claims (significantly extend the PDP)
2. Fully evaluate the claim against the policy outside of XACML
3. Split evaluation so that the PDP evaluates what can be expressed and evaluated in XACML while some other component evaluates the ABC4Trust specific parts of the policy and claim.

All have their drawbacks. The implementation costs of the first and the questions about the relevance of XACML raised by the second appear to make the third the least worst. Option 1, while attractive, appears infeasible, both in terms of implementation within the time and resources available and in terms of widespread acceptance. It would likely require an opening up of the ABC4Trust token (by an inspector) which may compromise user privacy. As user privacy is a keystone objective of the project this appears undesirable. Option 2 is also perhaps difficult to implement, given the difficulty in extracting information held by the PIP for use outside the XACML implementation. In option 3, one component would report its evaluation to the other, which would then make a final decision. Option 3, while not ideal form the perspective of a clean design, appears the most feasible alternative in terms of developing an actual implementation within the scope of the project. At the very least we partition the system such that each layer does what it has been built to do.

5 Does (should) the use of the presentation token by the service provider XACML system require inspector capabilities?

This has been discussed (to some extent) under issue 4, but is worth a more complete examination. The user will return an ABC4Trust presentation token to the service provider. The contents of this can only be used by an ABC4Trust aware component or after actual values are extracted by an inspector. Allowing inspection poses conceptual issues around user privacy, as discussed under issue 4. Not allowing inspection (and assuming no alterations to the XACML implementation to make it ABC4Trust ‘aware’) limits the subject attributes available to the PDP in making its decision to either what the service provider knows about the subject or what can be extracted from the token without inspector capabilities. Allowing inspection raises privacy issues as the service provider will have access to underlying subject attributes. This appears to be a fundamental issue. If we use XACML more or less as is, we either have inspector capability in the service provider or limit the information available to the PDP. An alternative would be to enhance the PDP (and SAML), increasing implementation costs.

If inspection capabilities where to be used the question arises where in XACML they should be placed. One option is the PDP. If so, then the presentation token will need to be passed to the PDP. If it is some other component (PIP, PEP, etc.) then the questions arises of how the attribute values extracted by the inspector are made available to the PDP. They could either by passed with the request until it reaches the PDP or injected into the databases maintained by the PIP.

Our conclusion is that the inclusion of inspector capabilities as standard to the processing of a user request invalidates user privacy and should be avoided.

6 How are the AuthN polices created?

The user needs to be provided with an AuthN (ABC4Trust) policy so that their ABC4Trust system can generate appropriate claim/evidence (in the form of a presentation token). Unless there are significant changes to the XACML implementation the XACML system of the service provider will store AuthZ (XACML) policies. The first issue in the provision of the AuthN policy to the user is whether they are:

a) Generated, possibly on demand, from the AuthZ policies, or,

b) Manually written and maintained.

This question should be relatively independent of the other issues identified in this section. Option a) avoids questions of data synchronization between the two policy stores. Option b) does not require the development of an automatic policy generator. The complexity of automatic policy generation, especially between two less than fully compatible systems, should not be underestimated. However, even if option b) is initially adopted, it should be fairly straightforward to integrate option a) at a later point.

A second question, for option b), is how is the relevant AuthN policy identified for each AuthZ policy? Note that AuthN policy generation takes place after the relevant AuthZ policy is located (by the PDP) based on the user request (R, R’). A fairly simple approach would be to store a reference with each AuthZ policy, identifying the relevant AuthN policy.

5 Analysis of Possible Integration Solutions

In this section we discuss three possible solutions for the integration of the authorization and authentication mechanisms. Each solution is analyzed in terms of the design issues identified in section 4, together with a discussion of the advantages and disadvantages of each. It is clear that multiple solutions are possible. And that the advantages/disadvantages depend on the view taken on the design issues identified. The first two approaches analyzed, the obligation and extraction approaches, were proposed in [5]. The third approach, the gateway approach, was developed during work on this deliverable. While not the design we propose it is useful to examine it in order to better understand the design options available.

5.1 Obligation Approach

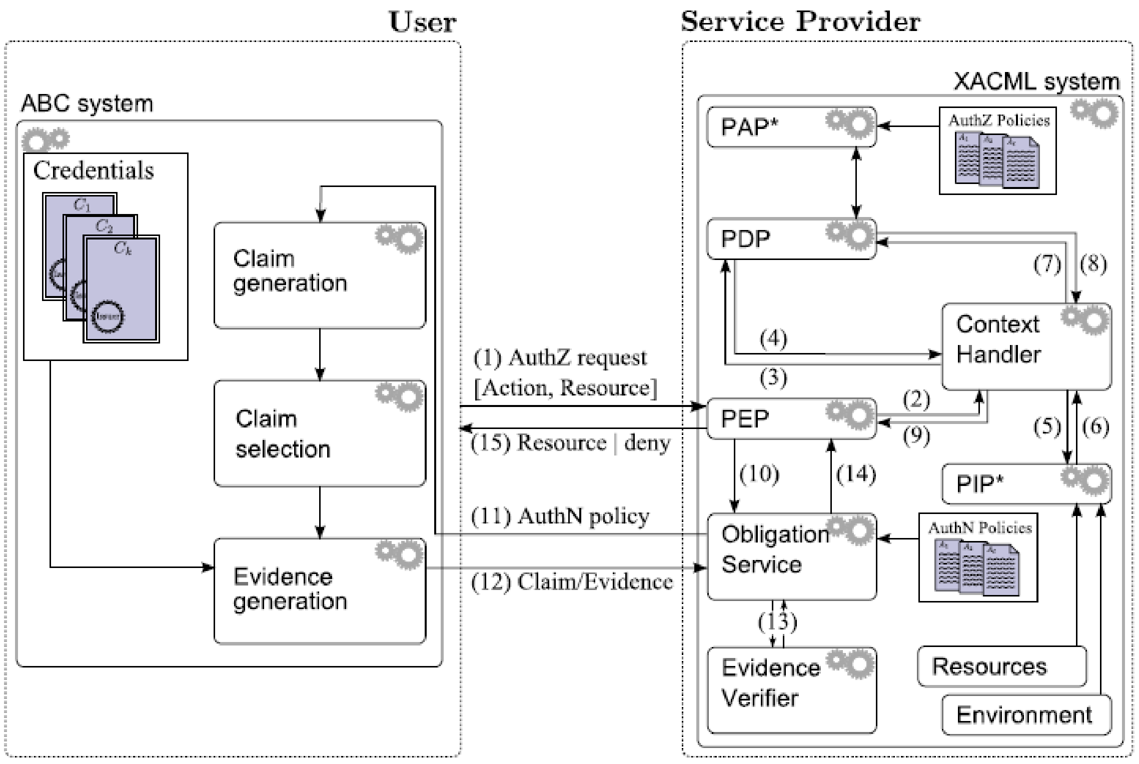

Figure 3: Obligation Approach from [5]

The obligation approach makes use of the Obligation Service of XACML to verify whether the claim/evidence supplied by the user satisfies the AuthN policy. This will obviously require significant changes to a standard implementation of the Obligation Service. An advantage is that the changes to the XACML implementation are limited to the Obligation Service only. However, the changes to the obligation service will be considerably large. It will need to understand ABC4Trust policies and presentation tokens and also to communicate directly with the user, in steps 11 and 12 from figure 3. The user's system will need to handle two avenues of communication, one with the PEP and one with the Obligation service. Alteration or extension of an XACML implementation will be needed to handle ABC4Trust entities on the service provider. The nested communication between user and service provider (steps 1, 11, 12 and 15 from figure 1) is relatively straightforward, even if it does mean that the two messages in each direction will likely have little in common. A more significant drawback is that the Obligation Service is making the final decision about whether access is granted or not. This can be seen in the flow of steps 12, 13, 14 and 15. This is an unusual function for the Obligation Service and also leaves the position of the PDP uncertain. The only PDP responsibility is deciding whether the initial request, unaccompanied by any claim/evidence, should be denied (unless the request is for a public resource). This minimization of the role of the PDP does not seem in accordance with the use of XACML. One could even question why XACML would be used at all if the roles of the PDP, PIP, etc. are so sidelined. It would appear that, in this approach, much of that functionality would need to be replicated in the Obligation Service.

In terms of the design issues identified in the previous section:

1 The AuthZ request is made before the AuthN policy is obtained. This allows the exchange between user and service provider to begin with a standard XACML resource request. Similarly, the final message (15) is a relatively standard allow/deny.

2 The Obligation Service supplies the policy. This is, as mentioned above, part of the significant changes required to the Obligation Service in this approach.

3 The structure of the message pairs is different. Steps 11 and 12 are not standard XACML, so they would need to be designed and implemented.

4 The Obligation Service checks whether the supplied claim/evidence satisfies the policy. Note that the supply of the AuthN policy, provision of the claim/evidence (step 12) and the check between that claim/evidence and policy all occur after the PDP has completed its work. As discussed in section 4, this raises questions as to the use of XACML in the design. Steps 2 to 9 appear to have no value other than to recover the AuthZ policy. This could be handled in a much simpler manner. Also, the extensions to the Obligation Service to fulfil this role would likely be considerable.

5 Inspector capabilities are not required. This is an advantage as user privacy is preserved. This does assume that the Obligation Service is provided with the ability to verify ABC4Trust policies. In effect, the Obligation Service either has, or has access to, significant portions of ABC4Trust functionality. This will be needed by any approach, as the ABC4Trust policy must be verified at some point. Otherwise the expressive capacity of ABC4Trust policies (beyond those of standard XACML policies) cannot be utilized.

6 In [5] AuthN policies are manually written and placed in a database for the Obligation service. However, it would be relatively easy to redesign this approach so that the AuthN polices are automatically derived from the AuthZ policies stored in XACML.

5.2 Extraction Approach

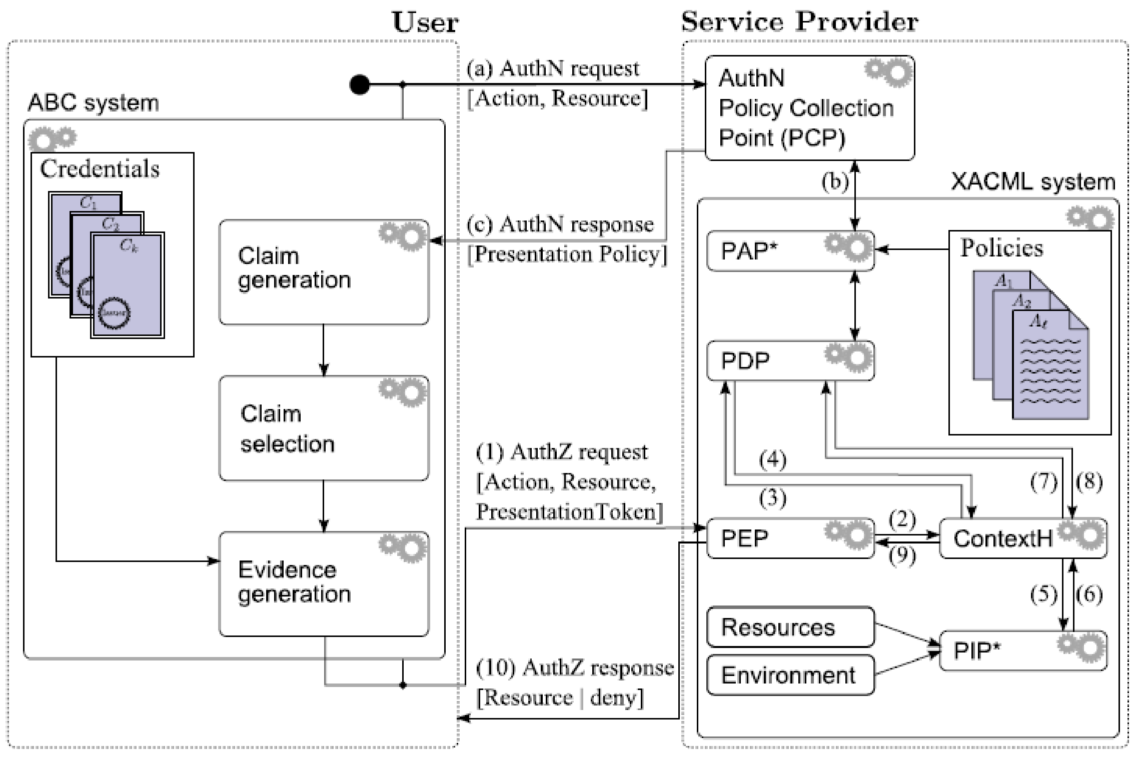
User
Service Provider
ABC system
Credentials
Claim generation
Claim selection
Evidence generation
(a) AuthN request [Action, Resource]
(c) AuthN response [Presentation Policy]
(1) AuthZ request [Action, Resource, PresentationToken]
(10) AuthZ response [Resource | deny]
AuthN Policy Collection Point (PCP)
(b)
XACML system
PAP*
Policies
PDP
PEP
ContextH
PIP*
Resources
Environment
(2)
(3)
(4)
(5)
(6)
(7)
(8)
(9)

Figure 4: Extraction Approach from [5]

The Extraction approach employs a two-stage message exchange, as opposed to the nested exchange of the obligation approach. First the user obtains the relevant policy for their request (in messages a, b, and c). Then, having generated an appropriate presentation token on the basis of that policy the user makes the AuthZ request (step 1). The actual evaluation of whether the presentation token satisfies the policy is handled by the PDP. This has both advantages and drawbacks. The advantage is that it allows the PDP to fulfil its normal role in XACML. The PDP is the component which is making the (final) decision as to whether the access request should be allowed and it is the only component making that decision. The XACML system, and the PDP in particular, appears much closer to fulfilling its normal role than in the Obligation approach, but the substantial changes required to a standard XACML approach to achieve this appear to be a considerable drawback The PDP must be able to evaluate ABC4Trust policies. Also the presentation token must either be provided to the PDP through the PEP and context handler or some other component, likewise ABC4Trust aware (perhaps the PEP) must extract the information provided in the presentation token and place into the databases maintained by the PIP. This does not refer to inspection capability, but to the information that the user normally reveals in a presentation token.

In terms of the design issues identified in the section 4:

1. The AuthN policy is obtained before the AuthZ request is made. While this is not a drawback in itself, it does require an extra (non-standard) interface into the XACML system, where the PCP (in step b) communicates with PAP. This will add to the implementation challenges.

2. A new component (the PCP) supplies the AuthN policy. This component would need to be designed and implemented.

3. The structure of the message pairs is different. The whole function of the PCP and its communication with the user, are additional to the standard XACML approach. This again appears to add to the implementation required.

4. The PDP checks whether the supplied claim/evidence satisfies the policy.

5. Inspector capabilities are not required. This is an advantage as user privacy is preserved; at least if the assumption is made that the PDP is able to verify ABC4Trust policies. As with the Obligation Service in the Obligation approach the PDP will either have, or have access to, significant portions of ABC4Trust functionality. As also noted above, some component on the service provider side will need this functionality.

6. In [5] AuthN policies are automatically derived (by the PCP) from the XACML policies. As with the Obligation approach it would straightforward to redesign this approach so that the AuthN policies were manually written and placed in a database for use by the PCP.

5.3 Gateway Approach

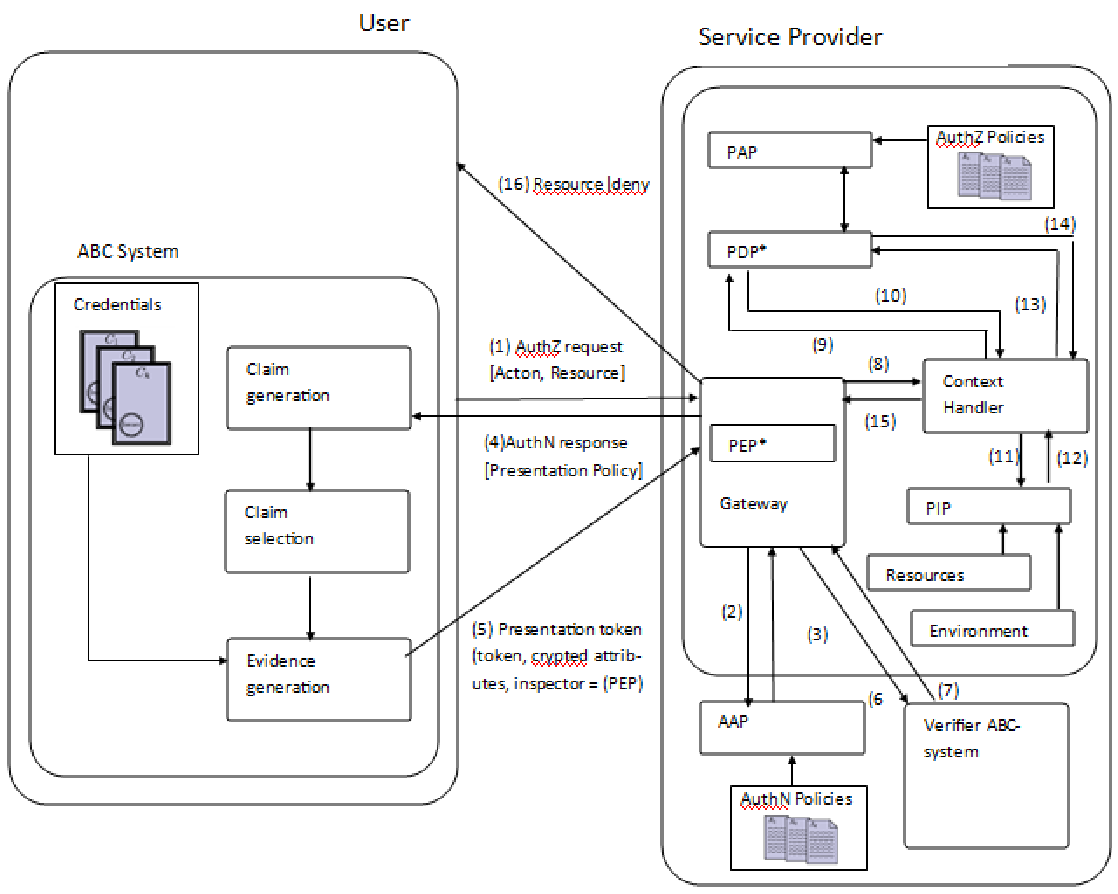

Figure 5: Gate way Approach

The gateway approach minimizes the changes required to an XACML implementation but at the expense of requiring inspector facilities. A gateway intercepts the initial AuthZ request (step 1). The gateway then uses another component (here called the AAP or Authorization-Authentication Processor, but conceptually similar to the PCP of the extraction approach) to retrieve the appropriate AuthN policy. The AuthN policy is returned to the user in step 4. The user can then make a second AuthZ request (step 5), now supplying a presentation token. Both pairs of messages (1/4 and 5/16) are similar to standard XACML request/response message, although with extra fields. Once the evidence is supplied (in step 5), the token is verified (steps 6 and 7) to ensure that the cryptographic proof supports the claims made. Inspection capabilities are then used to extract the underlying information from the token so that this information can be used by the PDP. This means that the PDP does not need to be ABC4Trust aware, nor do any other components of the XACML system. The drawback is that the use of inspection capabilities compromises user privacy. As an example, consider the following. The AuthN policy specifies that the user must be older than 18. The token makes this claim, with the actual age of the user only accessible using inspection facilities. A standard XACML PDP cannot evaluate a policy using an attribute predicate of this nature (Age >= 18). So the gateway would need to extract the actual age of the user from the token and make it available to the PDP. While this minimizes the changes to the XACML components required to implement this approach, the cost to user privacy appears a high price to pay.

Similarities between this approach and the extraction approach can be seen. The PCP in the extraction approach is fulfilling part of the role of the gateway. It would be straightforward to redesign the extraction approach with the PCP as a wrapper around the PEP in the form of the gateway. Similarly, the substantial internal changes to an XACML could be used in the gateway approach to avoid the issues around inspection. In summary, both approaches use a new component to supply the AuthN policy and both rely totally on the PDP to evaluate the claim/evidence versus the policy. The differences are whether that new component is relatively standalone or a wrapper around the PEP and whether XACML is made ABC4trust aware or inspection capabilities are used to avoid the need for this. The extraction approach uses the first of each of these two options, the gateway approach the latter of each two. Two more designs could be arrived at 1) by using a standalone component with inspection facilities and 2) by using a wrapper to supply the policy accompanied by extending the XACML components with ABC4Trust functionality. Inspection facilities, as discussed, appear an undesirable inclusion on the service provider. We return to discussion of the section option in the next section.

In terms of the design issues identified in the section 4:

1. The AuthZ request is made before the AuthN policy is obtained. As with the Obligation approach this allows the exchange between user and service provider to begin with a standard XACML resource request.

2. The AuthN policy is supplied by the gateway, by reference to the AAP. Note that the XACML system is not involved before the policy is returned to the user. This may result in some time/processing savings.

3. The message pairs (1/4 and 5/16) both use a (modified) XACML structure. This has advantages in terms of simplicity and implementation.

4. The PDP checks whether the supplied claim/evidence satisfies the policy. This has the advantage (similarly to the extraction approach) that the PDP fulfils its normal role in XACML, of policy checking. As discussed above, though, this will require inspection capabilities.

5. Inspection capabilities are required. This is a significant drawback of this approach. Routinely (i.e., for every request) unpacking the presentation token using inspection functionality represents a compromise, arguably fatal, to the intension of ABC4Trust.

6. The AuthN policies are manually written and placed in a database for access by the AAP. However, as with the two approaches examined previously, it would be relatively easy to redesign this approach. In this case so that the AuthN polices are automatically derived from the AuthZ policies stored in XACML.

6 Proposed Architecture

The three options discussed in the previous section all have their advantages. They also display significant disadvantages. The obligation approach sidelines most of the XACML system, except for the obligation service. However, it also requires significant changes to that service. The extraction approach requires significant changes to the entire XACML system. The gateway approach requires inspection capabilities, which is undesirable because of violation of user privacy, as already discussed.

In this section we propose an approach which avoids all these pitfalls. We acknowledge that it has drawbacks of its own, as will be discussed in the next section. However, as discussed in section 4, given the constraints on the current project, it appears infeasible to arrive at a solution without at least some drawbacks. It is simply a case of deciding which drawbacks raise the least problems. In our design we aim to give the primary and final responsibility for deciding access to the XACML system on the service provider. Access decisions are the primary function of XACML and so this decision flows naturally from the decision to employ XACML. However, it is clear from the above discussions that the service provider will require ABC4Trust functionality, in addition to XACML, if it is to issue and evaluate AuthN policies. Given the primary role we have given to XACML an attractive approach is to extend XACML with ABC4Trust functionality, while still retaining final decision making in the service provider's XACML system. This is similar to the final design possibility discussed under the gateway approach. However, instead of employing a gateway, the co-ordination between XACML and ABC4Trust on the service provider is placed in the PEP. This allows the PEP to maintain its role in XACML of being the point of communication with the user and also preserves XACML as the ultimate decision-making mechanism for access requests.

The proposed architecture is characterized mainly by a rather loose integration of XACML and the ABC4Trust authentication platform. The approach allows for, on the one hand, expressing a large fraction of practically relevant authentication requirements as part of standard XACML policies and thus reusing existing policies and, on the other hand, leveraging the full functionality of the ABC4Trust system in terms of advanced trust management for authentication transactions by expressing authentication aspects not expressible in XACML as ABC4Trust authentication policies. The approach leverages a modified standard XACML implementation as the AuthZ platform and leverages the full compatibility with XACML of large fractions of envisioned policies, while allowing for expressing more specific authentication requirements in ABC4Trust.

We next provide some more insight into this hybrid approach. The approach builds on expressing AuthN policies as part of AuthZ policies expressed in standard XACML for a large fraction of policies while keeping AuthZ policies expressed entirely in standard XACML. Specific authentication policy requirements not expressible through standard XACML can be accounted for by expressing them as ABC4Trust policies referenced from the AuthZ policies. These references will be generated as described below. The generated references – since they will be used as attribute names in XACML policies – should adhere to a naming convention so that they are fully distinguishable from each other and from "regular" attributes. This naming convention has to be an agreement between the AuthN and the AuthZ system (e.g., a prefixed naming with random or predicate-based suffix). The name of a reference attribute is unique to the predicate it refers to and is used for mapping the attribute to the predicate. The mapping between reference attributes is managed by the Auth^{2}Mediator component.

Concretely, AuthN policies or policy fragments communicated to the user as part of an authentication/authorization transaction can either be (1) dynamically derived from the AuthZ policy expressed in XACML or be (2) statically defined and referenced from the XACML AuthZ policy. The difference between dynamic and static reference attributes lies solely in how they are generated. They function identically when it comes to resolving user requests.

Approach (1) (dynamic generation) allows a large fraction of practically relevant policies to be handled in fully compliant XACML syntax and deriving the corresponding AuthN policies at run-time. Portions of the XACML policy which relate to user attribute authentication are handled in this way. This option is realized by replacing predicates in the AuthZ policy that cannot be evaluated in a privacy-enhanced manner by the XACML PDP with boolean attributes referred to as dynamic reference attributes. The replacement is done as pre-processing on the XACML policy, e.g., when the policy is loaded into the PDP at run-time. After replacement, only the boolean attribute is visible to the PDP and the predicate itself is handled by the AuthN system in a strong privacy-preserving manner.

Approach (2) is realized by including static reference attributes in XACML policies with the semantics that each such attribute refers to a statically defined AuthN policy. The XACML policy is authored with such static reference attributes already included to refer to the ABC4Trust policies. Each static reference attribute has the semantics that it is true if the corresponding ABC4Trust policy has been fulfilled by the user, in the same way as dynamically generated reference attributes. Static replacement can either make use of existing reference attributes or be added to the store managed by the Auth^{2}Mediator component as a new reference attribute. The static approach is particularly useful for policy components that can be expressed in the AuthN language but not the AuthZ language. The policy component only has to exist in the mapping held by the Auth^{2}Mediator component and is never expressed directly in the AuthZ policy (it is expressed, indirectly, by use of the reference attribute). Dynamic generation of the reference attribute is only possible with policy components that can be expressed in the AuthZ language.

Combining both options provides flexibility by reusing standard policies where appropriate and expressing powerful privacy-preserving authentication semantics where required in a single system. AuthN policies that can be expressed through the AuthZ policy language can be expressed in that language, while AuthN policies that cannot be expressed in the AuthZ policy language, are expressed as AuthN policies referenced from the AuthZ policy as described above.

For deriving AuthN policy fragments from AuthZ policies in Approach (1), standard XACML constructs are used, without any modifications. This allows for the reuse of any existing policy while, if required, mapping certain semantics of it to ABC4Trust AuthN policy fragments. Approach (2) builds on the attribute request construct of XACML that is used to refer to AuthN policies within AuthZ policies. This is done by using a static reference attribute and requiring its value to be true in case the referred-to policy is fulfilled by the user’s assertion.

The semantics of specifying a reference attribute is that the AuthN policy needs to be fulfilled by a presentation token of the user, that is, the user’s presentation token provided in response to a request needs to be verified successfully and the token needs to be successfully matched against the referred-to AuthN policy. The fulfillment of the AuthN policy can then be indicated to the XACML system by a boolean and hence be part of the evaluation of the AuthZ policy.

Overall, this approach has powerful semantics by deriving AuthN policies automatically for many common kinds of predicates, such as attribute requests or inequalities, and allowing for the fulfillment of any AuthN policy that can be expressed in the AuthN system handling user attributes, completely independent of XACML functionality. The fulfillment is formally captured as part of the XACML policy through a dedicated attribute.

6.1 Architecture

The detailed message flow of the proposed architecture is given in section 6.2. In outline:

1. The user makes an access request, in form of a regular XACML AuthZ request.
2. This is processed by the XACML system on the service provider. Usually a 'deny' will be returned, along with an AuthN policy which must be satisfied if access is to be granted. The only exception to this is if access can be granted without any information about the user (that is, it is a publicly accessible resource).
3. The AuthN policy applying to the request is composed by the system based on the applicable dynamically derived and statically defined AuthN policies and is returned to the user and employed by the user's ABC4Trust system for producing appropriate claim/evidence in the form of an ABC4Trust presentation token. Whether this is handled automatically or requires interaction with the user is a standard part of the ABC4Trust system. The claim/evidence is sent to the service provider, along with a repeated access request.
4. The claim/evidence is verified by determining whether associated the cryptographic elements support the exposed claims, as per the normal ABC4Trust process. This is the responsibility of an ABC4Trust system on the Service Provider and does not require inspection in the default flow as it uses the standard ABC4Trust approach. The AuthN system on the service provider verifies the parts of the policy against the claim which are dynamically derived or statically defined policies and provides this information to the PDP through the PEP via a standard XACML request. All attribute values provided in the claims are also part of the standard XACML request to the PDP.
5. A final access decision is made and communicated back to the user.

Note that step 2 will require extraction of the dynamic and static reference attributes from the AuthZ policy related to the request and translation from those to an AuthN policy (required by the user to prepare the appropriate claim/evidence) during evaluation time. This will require changes to XACML (so that the policy can be determined and returned) as opposed to the normal situation, where the policy goes no further than the PDP. It will also require translation of the AuthZ policy, respectively the extracted attribute references referring to dynamically derived and statically defined AuthN policies, into an AuthN policy. Step 4 requires an extension to the standard ABC4Trust system to allow for creating the XACML request towards the PDP. This will be discussed in section 6.2. The architecture is depicted in figure 6.

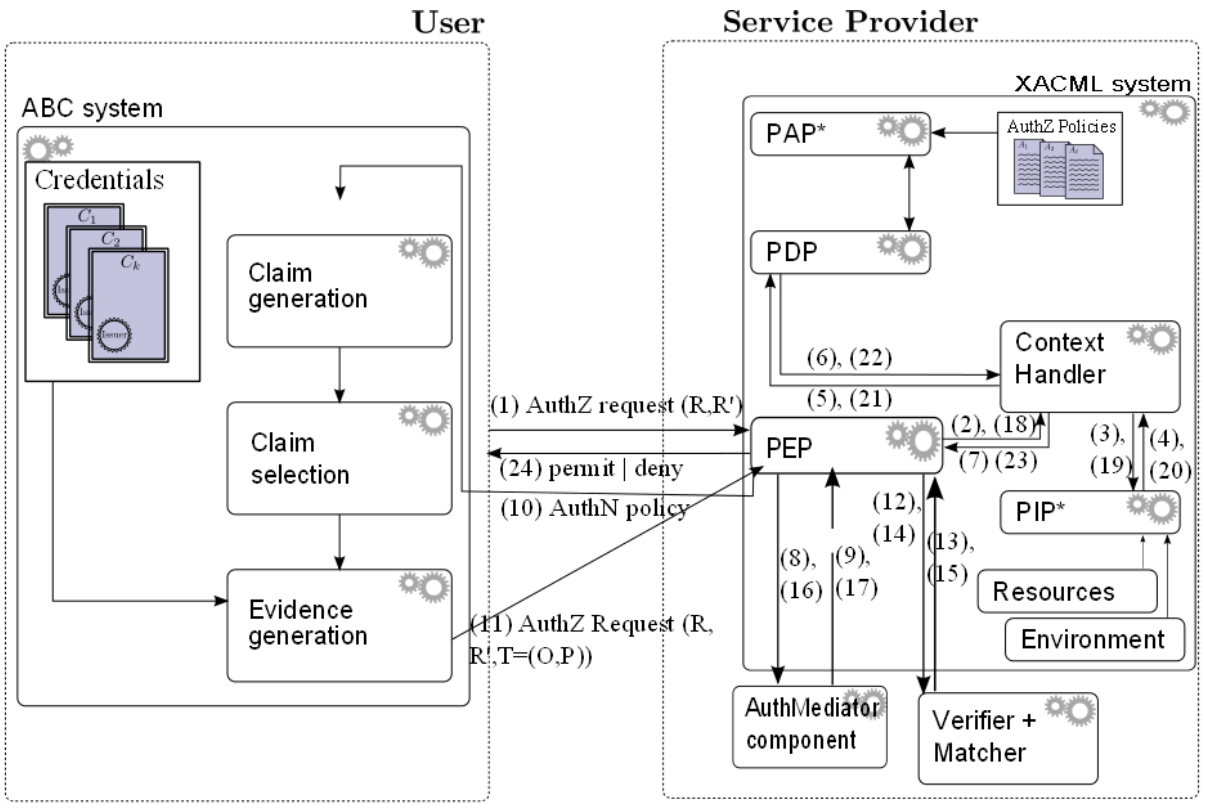

Figure 6: Proposed Architecture

6.2 Message Flow

The underlying assumptions for the message flow are that at request time a) the service provider knows nothing about the user b) the user knows nothing about the policies of the service provider. Thus a 2-round flow will be executed, that is, a flow comprising two request-response pairs.

Before this flow starts, a pre-processing of the XACML policy is required in which certain parts of the XACML policy that cannot be handled by XACML while at the same time preserving privacy are replaced with dynamic reference attributes. This replacement is explained in Section 6.4. In other words, the XACML policy now contains dynamic reference attributes in the place of certain predicates. Also, the policy may comprise static reference attributes since its definition.

<Message 1>
The user U makes an access request without a presentation token to a resource R and action R' of the service provider SP. (R, R') are expressed in proprietary form, e.g., the user clicking a link on a Web page.

<Messages 2-5>
The PEP creates an XACML request to the PDP and requests the PDP to compute a decision, where (R, R') are expressed in XACML request format. This is using the standard XACML flow through the Context Handler and PIP. We assume that nothing is known about the requesting user U, that is, there are no subject attributes contained in the XACML request to the PDP. The PDP computes a decision for the access request, which will be 'deny' unless (R, R') is a publicly accessible service/resource.
Additionally to computing the decision, the PDP derives the required reference attributes, both static and dynamic ones, representing the dynamically derived and statically defined authentication policies

relevant in the context of the requested access. This approach requires the PDP to be modified in order to capture the functionality of extracting the reference attributes for the AuthN policies which need to be fulfilled in order to make an access decision. More details about mapping AuthN and AuthZ policies in order to support dynamic derivation of attributes representing statically defined or dynamically derived authentication policies by the PDP are described in Section 6.3.

<Messages 6, 7>
The PDP returns “deny” as part of a standard XACML response and a data structure comprising attribute requests of dynamical and static reference attributes and “regular” attributes, expressed as a disjunctive normal form, that is, a disjunction of conjunctions of attribute requests. This data structure will be called A. Its semantics is that only one of the AuthN policies composed based on the minterms of this structure needs to be fulfilled by the user in order that the PDP be able to make an access decision. Obviously some policies will allow no choice (i.e., have only one minterm) but where such choice exists (e.g., the policy may essentially be in the form M_1 *or* M_2 *or* M_3 *or* …. *or* M_n, where M_i is a minterm and n is the number of minterms) the choice of which minterm to fulfil needs to be presented to the user.

<Messages 8, 9>
The PEP requests the Auth^{2}Mediator component to translate the structure A to an authentication (AuthN) policy A’ expressed in the standard ABC4Trust format, that is, an ABC4Trust presentation policy alternatives structure. More details about translation of AuthZ policies to AuthN policies are described in Section 6.4. The obtained AuthN policy comprises a list of alternatives, corresponding to the minterms, and the user is free to choose which alternative they will provide a claim against. Providing the choice of which of their attributes to reveal enables users to preserve their privacy to the maximum amount possible while still satisfying the service provider’s access policies. This supports the principles of user privacy and least privilege, as only the minimum information is released and the choice of which information to reveal is left to the user.

<Message 10>
The PEP sends A’ to the user in order to communicate to them which attribute assertions need to be released in order that an access decision can be made.
The user receives A'. Note that A’ is expressed in the vocabulary of the service provider in case different vocabularies are supported. In order to be able to process the request, the user’s system needs to map the policy A’ to its own vocabulary at this point. The translation between different vocabularies of the user and service provider (if they are different) is not addressed in this deliverable. This question is addressed in the deliverables focusing on ontology mapping. We simply assume that such translation is provided if it is necessary.

The user’s ABC4Trust system then computes claims candidates O_1, …, O_n, based on the user’s credentials, that can fulfill A’ (and, by extension, the corresponding minterms discussed above). One of these claims (or assertions) is chosen, let it be called O, either by the human user or automatically, or a combination of both. The user agent creates a cryptographic proof, or cryptographic evidence P for O, and O and P together form the presentation token T= (O, P).

<Message 11>
The user makes another access request to the resource R and action R’ of SP and attaches the presentation token T= (O, P) to this request. Note that this request is the same request originally made by the

user, however, with the presentation token T attached in addition to communicate certified attribute assertions about the user.

<Messages 12, 13>
The PEP receives T and asks the verifier (part of the ABC4Trust AuthN service at SP) to verify the cryptographic proof P of T w.r.t. the attribute assertion O of T.

Note that the cryptographic verification of the presentation token requires it to be expressed in the user’s vocabularies. That is, for support of ontology-based mapping between vocabularies of different parties, a mapping of the attribute assertion to the vocabulary of the service provider will be required after its cryptographic verification.

If the proof verifies, the presentation token description O can be considered authentic and can be used as trusted subject attribute information for computing the AuthZ decision. So the verifier indicates to the PEP that the assertion is valid and sends the set of attribute values and predicates included in O to the PEP.

<Messages 14, 15>
The PEP calls the AuthZ service of ABC4Trust for matching the presentation token description O against the presentation policy alternatives A’. Let matching == success express that O fulfills the policy A’. The semantics of this matching in case of success is that O fulfills one of the alternatives in the presentation policy. The index of this alternative is also returned as part of a success response. Note that while in figure 6 having both pairs of messages 12, 13 and 14, 15 may appear redundant; the calls 12 and 14 are to different services provided by the ABC4Trust system.

<Messages 16, 17>
If the matching was successful in the previous step, attribute values, including those for reference attributes (both static and dynamic) referring to ABC4Trust policy fragments are determined and translated to XACML syntax for representing attributes.

In this step, the ABC4Trust service identifies the attribute predicates corresponding to reference attributes that are fulfilled, determines the attribute values of revealed attributes, and determines the predicates that are undefined. The reference attributes corresponding to the predicates that are fulfilled are sent as boolean attributes with value true to the PEP in addition to the attribute values for released attributes as included in O.

<Messages 18 to 23>
The PEP constructs an XACML request from (R, R’) and the list of subject attribute values and attribute references sent by the ABC4Trust translator in the previous step. This is a straightforward step and requires only syntactical processing. We will refer to this XACML request as X. The PEP requests from the PDP a decision by making the XACML request X. Note that, with this message, the second round of the XACML decision process starts. This round is conceptually similar to the first one in that it is a resource access request.
The PDP computes the decision for the request X using standard XACML policy evaluation. This can at this point be “permit” or “deny”, depending on the attribute information provided. Note that it also can lead to another request for attributes in case the token provided was insufficient to compute an access decision, which can be the case when the user has not fully supplied information as requested in the AuthN policy.

The PEP may as an optional step inject the translated attribute assertion obtained from the user presented token into the PIP for use in future accesses.

<Message 24>
The decision by the PDP is enforced by the PEP and potentially the user receives access type R’ to resource R.

6.3 Mapping of authentication and authorization policies

There are multiple options for how the AuthN policies applying to an access request may be extracted from the system. These solutions range from handling a static table outside of the PDP mapping action and resources to authentication policies to dynamic policy extraction solutions requiring more changes to the PDP with different degrees of XACML compliance. Our objective is to propose an automated solution that is easily maintainable and that provides an acceptable level of XACML compliance.

Handling a static table outside of the PDP that maps resources and their associated actions to a set of special attributes representing authentication policies leads to a completely unchanged PDP. Such a solution has better standards compliance, however the table is manually created and difficult to maintain. Any change in the AuthZ or AuthN policies needs to be handled manually and with the consequent risks that updates to one will not be properly reflected in the other.

A dynamic solution where the PDP derives the special attributes representing AuthN policies that apply to (R, R’), in addition to computing the access decision, requires that the PDP be modified in order to capture the functionality of extracting the special attributes and reference attributes that need to be fulfilled in order to make an access decision. In to that, a link between the extracted AuthZ policy and the AuthN policy must be automatically generated. This avoids manual updates, those being difficult to maintain as already noted.

Mapping AuthN and AuthZ policies must start at building time, i.e., when AuthZ policies are authored (see figure 7). XACML Authorization policies are created by an administrator through the policy administration point (PAP), stored in the PRP and loaded into the PDP. New policies loaded into the PDP are automatically analyzed by a specific algorithm that extracts XACML conditions related to each rule in order to transform them into predicates. This algorithm will be written and implemented as part of a later deliverable. The predicates that are extracted from XACML policies are then sent to a policy mediation module that gives them references in the authentication system and sends these references back to the to the PDP. The PDP replaces the XACML conditions by the received references.

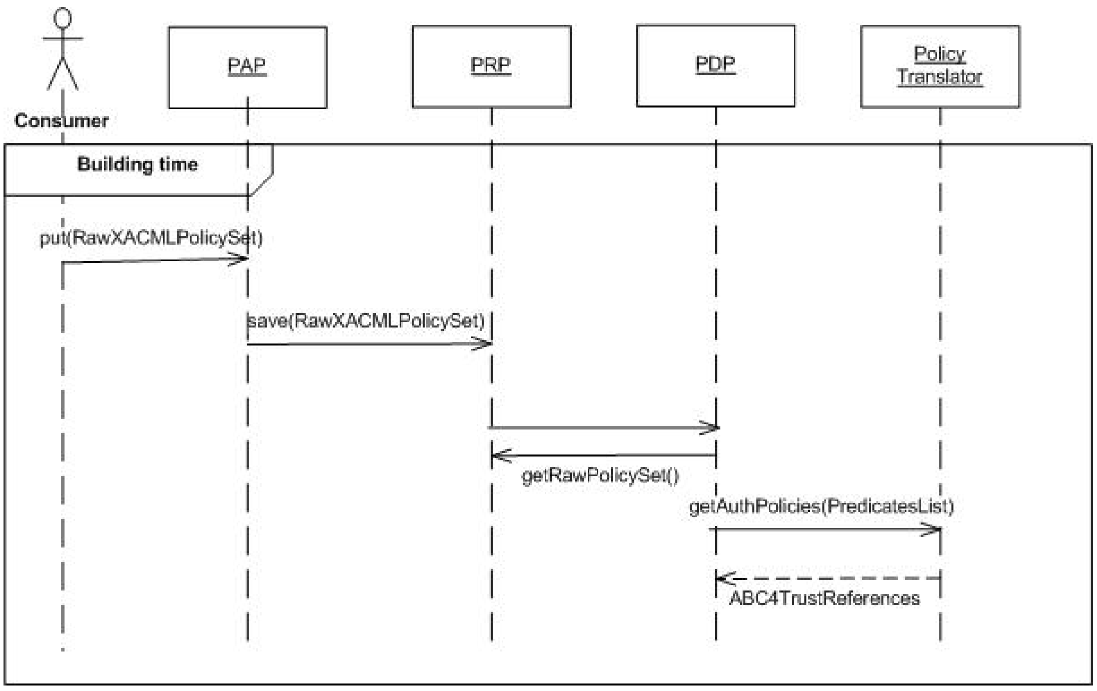

Figure 7: Linking AuthZ and AuthN policies with a reference

6.4 AuthN/AuthZ Policy translation

As described in the previous section, identification and extraction of predicates is done as an augmentation of the XACML system. Next, we discuss the details of the translation from a standard XACML policy (the AuthZ policy) to an ABC policy (the AuthN policy). After the authentication flow, a translation from the authenticated attributes (supplied by the user) to a standard authorization request expressed in XACML is also needed.

We call the component which performs all these tasks the Authentication-Authorization Mediator (Auth^{2}Mediator) component (see figure 6), as the task is more complex than performing a simple translation between two languages.

The Auth^{2}Mediator has three main functions:

1. creation and maintenance of a mapping between reference attributes (either statically or dynamically generated) and the parts of XACML rules (such as predicates over attributes) that they replace;
2. composition of AuthN policies from a disjunctive normal form (DNF) of both dynamic and static reference attributes, the DNF of which has previously been created from the XACML rules;
3. composition of the core part of an XACML request based on the output of the authorization part of the authentication system and the corresponding DNF from the former request.

6.4.1 Mapping

When the XACML system identifies an element it cannot itself evaluate in a privacy-preserving manner in a rule, e.g., an inequality predicate over an attribute from a type with a total order defined on it, it has to replace it with a reference to a corresponding (partial) ABC4Trust policy. The reference is an attribute of type boolean with its name allowing it to be mapped to the replaced element. To realize this, a mapping is needed between the identified predicates and the corresponding policies. This is the first task of the Auth^{2}Mediator component.

When a request comes in to the Auth^{2}Mediator component with the semantics of creating a reference for a given predicate, the component acts in the following way:

1) it creates a canonicalized version of the given predicate;
2) it performs a lookup in its internal store based on the result of the canonicalization:
    a. if the lookup does not return a value, it creates a new entry in the internal data store from the canonicalized version of the to-be-mapped predicate to a freshly created reference. Also, it creates a (partial) ABC policy from the predicate and creates a mapping from the reference to the newly created policy, then returns with the fresh reference;
    b. if the lookup's result is an already existing reference, then it returns with that.

The system which invoked the request can handle the reference as it pleases (in the proposed architecture, the XACML system will replace the identified predicate in the XACML policy with the reference).

The canonicalization of the predicates is an internal task of the Auth^{2}Mediator, it remains internal hence the representation does not have to be interpreted outside of the component. The internal storage can be any suitable solution (e.g., key-value store, RDBMS, either in-memory or on-disk implementation). The canonicalization step is necessary and it has to be a proper canonicalization, since it is the essential part of the mapping and ensuring that the same XACML elements (predicates) are mapped to the same reference attributes.

6.4.2 Composition of AuthN policies

When a request arrives at the service provider from a user to access a resource/service/etc., this request is – as discussed earlier – processed by the AuthZ system of the service provider, which, beside the "Deny" decision, will return an AuthZ policy which has to be fulfilled if access to the resource is to be granted. This AuthZ policy has to be translated into an AuthN policy which then the user can provably fulfill.

This translation is mainly based on the mapping function described earlier and on a disjunctive normal form (DNF) which is derived from the AuthZ policy. This DNF comprises the corresponding reference attributes from the AuthZ policy, the references of which were acquired from the Auth^{2}Mediator component earlier as well as reference attributes referring to statically defined policies. The DNF has semantics of the user being required to fulfill the authentication policy corresponding to one of the disjuncted conjunctions of reference attributes of the DNF (i.e., one of the minterms discussed above). Based on the reference attributes in the DNF, an AuthN policy can be created by the mediator by transforming each minterm of the DNF into a presentation policy. This is done by looking up the corresponding partial ABC policies for the reference attributes and composing these into a presentation policy. The list of presentation policies for the full DNF is composed to a so-called "presentation policy alternatives" data structure of the ABC4Trust system (cf. [4]).

In the scope of the project, the ABC policies behind the reference attributes are not arbitrarily complex authentication policies but rather simple ones as they correspond to the replaced XACML elements such as predicates, hence composing them into a full ABC policy is achievable even in the absence of a complete policy composition algebra.

6.4.3 Composition of XACML requests

After the authentication protocol has been successfully completed (i.e., the presented cryptographic evidence indeed implies the given claims (the attribute-value pairs)), the authenticated attribute-value pairs have to be translated into a proper XACML request or core part thereof.

For this translation two things are needed: the DNF which was the basis of the authentication flow and the results of the authentication flow, that is, a “presentation token description” element and an index identifying which minterm was fulfilled in the DNF. Given these items, the Auth^{2}Mediator component composes an XACML request based on the following basic principles:

i. it first identifies the minterm in the DNF based on the index;

ii. for every reference attribute in the given minterm, it marks the reference attribute as True; for every “regular” disclosed attribute it sets it to the given value of the presentation token;

iii. every other attribute will remain undefined in the computed XACML request.

Following these rules results in an XACML request which fulfills at least one minterm of the DNF. Thus, evaluation results for every reference attribute and values for all “regular” attributes that have been disclosed through the presentation token are provided to the XACML system, meaning that all (reference) attributes needed by the PDP for making a decision for this access are available. The attributes which are part of other minterms remain undefined in the XACML request. This is the natural approach for handling reference and other attributes about which either nothing is stated by the user in the presentation token or which are only part of other minterms of the DNF. This also avoids certain complications related to logical implications between predicates corresponding to reference attributes in the fulfilled minterm and other parts of the DNF.

6.5 Vulnerability Analysis

The integrated AuthZ/AuthN system essentially comprises three modules, namely, the identity provider, the service provider, and a protected resource. When the user supplies the Service Provider with a claim/evidence generated by its authentication system in response to the AuthN policy supplied by the service provider, the decision on granting or denying access to the requested protected resource is made based on the resource type, action requested and the user privileges. The ABC4Trust system will manage user credentials and will facilitate generation of claims to be sent forth to the authentication system hosted within or outside the service provider. Cryptographic evidence is generated by the ABC4Trust system on behalf of a user and appended to the user's request for accessing the protected resource. The PEP at the service provider's end calls the ABC4Trust service for matching the user's presentation token description O against a presentation policy A’. If the token fulfills the policy A', a successful match is ascertained. The following exploits against the AuthN/AuthZ system will be tested (not necessarily in the order listed):

- _ Certicate forgery
- _ Spoofed IP addresses of the identity provider and the service provider

- _ Man-in-the-middle attacks
- _ Metadata Poisoning/Enumeration
- JavaScript Injection
- _ Cookie Modification/Tampering
- _ Cross Site Scripting
- _ Cross Site Request Forgery
- _ SSL vulnerabilities (against TLS v1.0)
- _ Tomcat Management Bugs
- _ Communication vulnerability exploit for 'protected resources' i.e., hidden data
- _ Traffic analysis (encrypted or otherwise)

Communication between the client and the service provider is facilitated through RESTful web services, and therefore, relevant exploits will be run to test the resilience of the architecture against penetration attempts through the deployed RESTful services. Further details on the procedures adopted to run each of the above exploits against the AuthZ/AuthN system are provided in the work package WP6.1.1 document.

7 Conclusion

The architecture proposed in section 6 has a number of advantages over the options considered in section 5. Primary amongst these is a consistent emphasis on simplicity. This can be seen in the communication between user and service provider, locating communication between XACML and the additional features on the service provider in the PEP and comparatively low implementation effort required.

In terms of the design issues identified in the section 4:

1. The AuthZ request is made before the AuthN policy is obtained. This allows the exchange between user and service provider to begin with a standard XACML resource request.

2. The AuthN policy is directly supplied to the user by PEP. This allows all communication between the user and the service provider to be through the PEP, as in standard XACML.

3. The message pairs (1/10 and 11/24) both use a (modified) XACML structure. This has advantages in terms of simplicity and implementation.

4. This responsibility is shared between the PDP and the ABC4Trust system on the user side. The issues around this were discussed in section 4.

5. Inspection capabilities are not required.

6. AuthN policies are derived from information supplied by the PDP. This is used by a translation component (the Auth^{2}Mediator component) on the service provider to form the policy that is returned to the user. We consider this design to be agnostic (as with the other designs) whether this is automated or relies on pre-authoring.

The design proposed requires relatively limited changes to XACML when compared to the obligation approach and, especially, the extraction approach. This has benefits in terms of both resources required for system development in the likelihood of system acceptance. Similar comments apply when the symmetrical nature of the messages between user and service provider in the proposed design are compared to asymmetrical nature of extraction and obligation approach. While the gateway approach also minimizes changes to XACML, it does so by requiring inspection functionality, which is not required in the proposed architecture. The proposed architecture does split responsibility for policy evaluation between the PDP and the service provider's ABC4Trust system, this cannot be avoided if changes to the XACML implementation are to be minimised, as discussed in section 4.

The architecture proposed here is very generic in terms of trust management and authentication functionality because it can leverage the full syntax and semantics of ABC4Trust in the statically defined AuthN policies referenced from AuthZ policies through special attributes. This is possible by keeping the AuthN policy in the ABC4Trust layer, while referring to it in the AuthZ layer. Despite the simplicity, the full AuthN policy is abstractly yet formally represented in the AuthZ policy through the dynamic and static reference attributes. Once the verification of the presentation token has succeeded, it implies AuthN policy fragments are satisfied and so the reference attributes can be considered to be true. These Boolean-valued reference attributes are sent to the PDP together with any other the subject attributes from the presentation token that can be safely handled by the PDP. The attributes of the AuthN policy can be provided to the AuthZ system and allow for all functionality of relating those subject attributes to object attributes. Thus, it is possible to use the full XACML functionality in the policy evaluation.

When it comes to dynamically derived attributes, one can author XACML policies in the unchanged AuthZ language and have AuthN policies derived automatically, without the need of keeping them manually synchronized. This particularly allows for covering a large spectrum of envisioned use cases with standard XACML policies and revert to statically defined AuthN policies for more special requirements.

Overall, the approach can be seen as combining the full functionality of ABC4Trust or other supported AuthN systems, while at the same time allowing the retention of the full XACML functionality. Only where the AuthN and AuthZ need to interface, that is, attribute information being communicated from the AuthN to the AuthZ layer, the functionality is reduced to the intersection of both.

The approach provides for a functional partitioning according to the needs for an extensible system architecture. Concretely, the functionality that is standard XACML is handled by XACML, while mechanism-specific functionality is handled by the respective authentication mechanism, while still allowing to have XACML semantics for the parts of the functionality of the AuthN system that XACML supports, e.g., relating subject to object attributes. This approach is scalable in terms of allowing for any number of different AuthN mechanisms in a single deployment. In our intended pilot deployments within the project, we will use a single Privacy-ABC mechanism only.

The approach also composes with many other AuthN schemes besides ABC4Trust. For simpler schemes such as SAML, the standard XACML approach of existing bindings can be used, while, for more advanced approaches, the advanced semantics can be handled within the AuthN layer without complicating XACML, as presented. Thus, each new technology requires a more or less elaborate AuthN support extension to the AuthZ framework. This approach keeps different AuthN technologies completely independent and also does not require method-specific changes to the major XACML components.

Compared with some of the other approaches discussed in section 5, this approach is substantially better suited for the purposes of ABC4Trust for the above reasons. Also, the almost full compatibility with the XACML standard in terms of components is a major advantage for practical deployments as the architecture can handle both our new privacy-enhancing protocols as well as legacy protocols such as SAML as used today.

Bibliography

[1] J. Lopez, R. Oppliger and G. Pernul, "Authentication and authorization infrastructures (AAIs): a comparative survey," *Computers & Security,* vol. 23, no. 7, pp. 579-590, 2004.

[2] P. Bichsel, J. Camenisch and F. Preiss, "A comprehensive framework enabling dataminimizing authentication," in *Proc. of the 7th ACM DIM,* 2011.

[3] OASIS, "eXtensible Access Control Markup Language (XACML) Version 3.0," 2013. [Online]. Available: http://docs.oasis-open.org/xacml/3.0/xacml-3.0-core-spec-os-en.pdf. [Accessed 18 August 2014].

[4] "ABC4Trust," ABC4Trust, [Online]. Available: https://abc4trust.eu/. [Accessed 18 August 2014].

[5] D. Ayed, P. Bichsel, J. Camenisch and J. den Hartog, "Integration of Data-Minimising Authentication into Authorisation Systems," in *Trust and Trustworthy Computing, 7th International Conference, TRUST 2014*, Heraklion, Greece, 2014.

[6] J. Camenisch, I. Krontiris, A. Lehmann, G. Neven, C. Paquin, K. Rannenberg and H. Zwingelberg, "Architecture for Attribute-based Credential Technologies," ABC4Trust, [Online]. Available: https://abc4trust.eu/download/ABC4Trust-D2.1-Architecture-V1.2.pdf. [Accessed 14 October 2014].