Digital Security: Cybersecurity, Privacy and Trust
DS-02-2014 Access Control

European Commission | Horizon 2020 European Union funding for Research & Innovation

From Real-world Identities to Privacy-preserving and Attribute-based CREDentials for Device-centric Access Control

WP5– Attribute-Based Access Control

Deliverable D5.1 "Specification and Initial Design of the ABAC Infrastructure"

Editor(s):	Giuseppe Bianchi (CNIT), Alberto Caponi (CNIT), Claudio Pisa (CNIT), Luigi Stammati (CNIT)
Author(s):	Giuseppe Bianchi (CNIT), Alberto Caponi (CNIT), Claudio Pisa (CNIT), Luigi Stammati (CNIT), Tooska Dargahi (CNIT), Michael Sirivianos (CUT), Yiannis Charalambous (CUT), Savvas Zannettou (CUT), Christoforos Ntantogian (UPRC), Nikolaos Vavoulas (UPRC), Faidon Lalagianis (UPRC), Eleni Veroni (UPRC), Christos Xenakis (UPRC), Vangelis Bagiatis (UPCOM), George Savvas (UPCOM), Yannis Katsaros (EXUS), Anuj Sharma (EXUS), Antonis Hatzikonstantinou (EXUS), George Gugulea (CSGN), Mihai Togan (CSGN)
Dissemination Level:	PU – Public
Nature:	R
Version:	1.2

ReCRED Project Profile

Contract Number	653417
Acronym	ReCRED
Title	From Real-world Identities to Privacy-preserving and Attribute-based CREDentials for Device-centric Access Control
Start Date	May 1st, 2015
Duration	36 Months

Partners

University of Piraeus	University of Piraeus research center	Greece
Telefónica Investigación y Desarrollo	Telefonica Investigacion Y Desarrollo Sa	Spain
verizon	Verizon Nederland B.V.	The Netherlands
certSIGN BY UTI	Certsign SA	Romania
wedia	Wedia Limited	Greece
EXUS	EXUS Software Ltd	U.K.
upcom Bringing business and IT together	Upcom Bvba (sme)	Belgium
THE PRODUCTIZERS	De Productizers B.V.	The Netherlands
	Cyprus University of Technology	Cyprus
	Universidad Carlos III de Madrid	Spain
cnit	Consorzio Nazionale Interuniversitario per le Telecomunicazioni	Italy
BAKER & MCKENZIE	Studio Professionale Associato a Baker & Mckenzie	Italy

Document History

Version	Date	Author	Remarks
0.1		Alberto Caponi (CNIT)	Initial Table of Contents
0.2		Claudio Pisa (CNIT)	Modified Table of Contents
0.3 - 0.16		Claudio Pisa (CNIT)	All deliverable authors contribution merging
0.17		Claudio Pisa (CNIT)	Review merging
0.18		Claudio Pisa (CNIT), Alberto Caponi (CNIT)	Reference and cross-reference check
0.19-0.20		Claudio Pisa (CNIT)	Document layout and format check
1.0-1.1		Claudio Pisa (CNIT)	Final versions
1.2		Claudio Pisa (CNIT)	Changes to reopened deliverable

Executive Summary

This document is part of the WP5 – Attribute-Based Access Control of the ReCRED project. The purpose of this deliverable is to report the definition and the description of the initial specification and design of the Attribute-Based Access Control (ABAC) architecture. The design of the architecture starts from an investigation on the state of the art of ABAC platforms that should be integrated and deployed in the ReCRED framework.

Since the ReCRED project has a special regard for the privacy of online user’s identities, we payed particular attention to anonymous credential systems like Idemix and U-Prove with the final target of integrating and deploying such technologies in the ReCRED framework. As reference for the integration we considered the results of the ABC4Trust and FIWARE projects. Last but not least, since the ReCRED project aims to provide Device Centric Authentication (DCA) together with ABAC features, we investigated also technologies that integrate ABAC inside the user’s devices like the IRMA project and discussed the implementation of ABAC technologies in a secure and trusted environment (i.e. Trusted Execution Environment) inside the personal device.

Moreover, the document provides a description of which attributes and policies will be involved in the ReCRED procedures for the Attribute Based Access Control and the format to describe them. Finally, the document provides a discussion of advantages that such ABAC architecture should bring inside real application scenarios like: financial services, campus Wi-Fi and services, age verification and discount for ISIC students.

The document is organized as follows. Chapter 1 provides a description of the Access Control mechanisms and motivations for using Attribute Based Access Control. The discussions of the state of the art for Attribute Based Access Control technologies and architectures, initial prototyping of ABAC components and the integration between them are discussed in Chapter 2. The definition of attributes and policies to be used for ABAC and format specification for them are reported in Chapter 3. The application scenarios of the ReCRED project and the benefit of using ABAC on them are reported in Chapter 4. To conclude the deliverable, security and privacy considerations on the technologies discussed in the document are discussed in Chapter 5, while our final general conclusions are included in Chapter 6.

Table of Contents

List of Figures

1 Introduction

The main target of the ReCRED project is to design and implement an architecture that allows a user to simplify online identities management by consolidating them and by exploiting the concept of *Device Centric Authentication (DCA)*. However, another main objective of the project is to go beyond the state of the art of Device Centric Authentication platforms by integrating privacy-aware access control capabilities, that are lacking in actual DCA platforms. Indeed, this is a natural step since user identities are basically formed by user attributes that can be exploited in order to realize an Attribute Based Access Control (ABAC) on resources that the user wants to access. The Attribute Based Access control architecture of the ReCRED project is mainly focused on the **integration** and **deployment** of the state of the art of anonymous credentials platforms such as Idemix and U-Prove, in order to become a referring platform ready for the market and industrial exploitation. Moreover, the ABAC architecture will maintain the main device centric approach of the project thanks to the integration of such ABAC systems in the user's device, thanks to the implementation on trusted execution environments (TEE) of devices. To reach such target, ReCRED project is investigating on existing and open source TEE platforms in order to provide a full and secure implementation of ABAC technologies directly on the device.

1.1 Attribute-Based Access Control

One of the purposes of the authentication of users requesting to perform an action in a system (e.g. requesting or modifying a resource) is the effective verification of the permissions that such user has with respect to the specific requested action. Indeed, this is not the main scope of authentication systems but it is one step forward demanded to check the authorization on actions that the user is allowed to perform. Access control is one of the functionalities that enable to perform the verification of authorizations of a user when requesting to perform an operation over a resource.

Traditionally, access control has been based on the identity of a user or by pre-defined attribute types such as roles or groups assigned to that user. This approach to access control is often cumbersome to manage, since it requires to associate capabilities (authorized operations) directly to users or their roles or groups. Moreover, the requester qualifiers of identity, groups, and roles are often insufficient in the expression of real-world access control policies. An alternative is to grant or deny user requests based on arbitrary attributes of the user and arbitrary attributes of the resource, and environment conditions that may be globally recognized and more relevant to the policies at hand, that takes the name of *Attribute Based Access Control* (ABAC). ABAC enables resource and service providers to apply an access control policy without prior knowledge of the specific requester and for an unlimited number of users that might require access. As a new user joins the system, rules and resources do not need to be modified. Since the user is assigned the attributes necessary for access to the required objects no modifications to existing rules or object attributes are required.

The evolution of access control models starts from the definition of *Mandatory Access Control* (**MAC**) and *Discretionary Access Control* (**DAC**) [1][2]. Such kind of access control was first implemented in the U.S.A. Department of Defense (DoD) applications and often employed in government and military facilities. Mandatory access control works by assigning a classification label to each file system object. Classifications include confidential, secret and top secret. Each user and device on the system is assigned a similar classification and clearance level. While it is the most secure access control setting

available, MAC requires careful planning and continuous monitoring to keep all resource objects' and users' classifications up to date.

As the highest level of access control, MAC can be contrasted with lower-level discretionary access control (DAC), which allows individual resource owners to make their own policies and assign security controls.

As networks and systems grew, the need to limit access to specific protected objects spurred the growth of *Identity-Based Access Control* (**IBAC**) that employs mechanisms such as access control lists (ACLs) to capture the identities of those allowed to access the object. If a subject presents a credential that matches the one held in the ACL, the subject is given access to the object. Individual privileges of the subject to perform operations (read, write, edit, delete, etc.) are managed on an individual basis by the object owner. Each object needs its own ACL and set of privileges assigned to each subject. In the IBAC model, the authorization decisions are made prior to any specific access request and result in the subject being added to the ACL. For each subject to be placed on an ACL, the object owner must evaluate identity, object, and context attributes against policy governing the object and decide whether to add the subject to the ACL. This decision is static and a notification process is required for the owner to re-evaluate and perhaps remove a subject from the ACL to represent subject, object, or contextual changes. Failure to remove or revoke access over time leads to users accumulating privileges.

The most important improvement on the management of permissions and access control policies is provided by the *Role-Based Access Control* model (**RBAC**) [3][4][5] that employs pre-defined roles carrying a specific set of privileges associated with them and to which subjects are assigned. For example, a subject assigned the role of Manager will have access to a different set of objects than someone assigned the role of Analyst. In this model, access is implicitly predetermined by the person assigning the roles to each individual and explicitly by the object owner when determining the privilege associated with each role. At the point of an access request, the access control mechanism evaluates the role assigned to the subject requesting access and the set of operations this role is authorized to perform on the object before rendering and enforcing an access decision. Note that a role may be viewed as a subject attribute that is evaluated by the access control mechanism and around which object access policy is generated. As the RBAC specification gained popularity, it made central management of enterprise access control capabilities possible and reduced the need for ACLs.

ACLs and RBAC are in some ways special cases of *Attribute-Based Access Control (***ABAC**) in terms of the attributes used. ACLs work on the “identity” attribute while RBAC works on the “role” attribute. The key difference with ABAC is the concept of policies that express a complex *Boolean* rule set that can evaluate many different attributes. While it is possible to achieve ABAC objectives using ACLs or RBAC, demonstrating access control requirements compliance is difficult and costly due to the level of abstraction required between the AC requirements and the ACL or RBAC model. Another problem with ACL or RBAC models is that if the AC requirement is changed, it may be difficult to identify all the places where the ACL or RBAC implementation needs to be updated.

Trying to implement IBAC or RBAC access control decisions would require the creation of numerous roles that are ad hoc and limited in membership, leading to what is often termed “role explosion”. A method is needed to make access control decisions without previous knowledge of the object by the

subject or knowledge of the subject by the object-owner. By relying upon the concepts of subject and object attributes consistently defined between organizations, ABAC avoids the need for explicit authorizations to be directly assigned to individual subjects prior to a request to perform an operation on the object. Moreover, this model enables flexibility in a large enterprise where management of access control lists or roles and groups would be time consuming and complex. Leveraging consistently defined attributes that span both subjects and objects, authentication and authorization activities can be executed and administered in the same or separate infrastructures, while maintaining appropriate levels of security. In general, ABAC avoids the need for capabilities (operation/object pairs) to be directly assigned to subject requesters or to their roles or groups before the request is made. Instead, when a subject requests access, the ABAC engine can make an access control decision based on the assigned attributes of the requester, the assigned attributes of the object, environment conditions, and a set of policies that are specified in terms of those attributes and conditions. Under these arrangement policies can be created and managed without direct reference to potentially numerous users and objects, and users and objects can be provisioned without reference to policy.

1.2 Integration of ABAC in ReCRED

The state of the art of *Device Centric Authentication* (DCA) platforms does not provide such access control capabilities since they are only focused on providing authentication methods to the user. The ReCRED platform will fill this gap by integrating support for attribute-based access control (ABAC) allowing to support the validation of the complete identity of an individual when the verifier considers only a specific identity attribute in order to grant the individual access to a resource. The ReCRED architecture for ABAC mainly addresses the fragmentation of the access control models and aims to create a platform that is open, i.e., designed and developed in such a way that it may interact with all the existing and new emerging standards and solutions in the area of user identification, authentication and access management.

A key goal of the entire ReCRED platform is to guarantee the protection of the privacy of the individuals using the services. In order to match such goal, the ReCRED ABAC architecture adopts a private credentials approach to let users prove their identity attributes safely from their device to the online or physical relying service. Indeed, in most of the real-world scenarios, users do not need to reveal their complete identity as the verifiers require knowing only an aspect of their identity (e.g., their age, their home address, their profession, whether they are students, etc.). Yet, users are forced to reveal their credit card details or present ID cards. By facilitating attribute-based access control, our solution becomes privacy-preserving by design thanks to the integration of *Anonymous Credential Systems*, i.e. cryptographic protocols that can support attribute-based access control (ABAC) like Idemix [6] and U-Prove [7].

In order to provide a fast implementation and deployment of the ABAC framework, we gave particular attention to the results of relevant EC-funded projects like ABC4Trust [8] and FIWARE [9] to be integrated and deployed as a baseline architecture for the privacy-preserving ReCRED ABAC infrastructure.

2 Attribute-Based Access Control Architecture

The ReCRED Device-Centric Reference Architecture described in Deliverable D2.3 and partially depicted in Figure 2 includes sub-components that are able to issue, verify, store and manage credentials.

Authorities such as Universities as well as the Identity Consolidation Service (described in Deliverable D4.1) can issue credentials to users that attest information about them in the form of attributes. When accessing online resources, users can select on their devices the attributes that they want to disclose, and whether they want to reveal their identity as well. Online Services that trust the issuing authorities can verify these attributes, check them against their policies, and decide to give access to resources.

This architecture allows for i) privacy preservation: the user can choose which attributes to disclose to each service; ii) device-centric credential management: credentials are stored securely on the user device and connecting to the credential issuer is not needed in the general case; iii) attribute-based policies: online services can define general policies that follow their security model.

The ReCRED architecture can support different underlying attribute-based access control protocols: Idemix and U-Prove but will be extended to other protocols (e.g. ABE-based) during the project. This support is achieved through a protocol agnostic common interface. Indeed, the ReCRED ABAC architecture conforms to the FI-WARE Privacy Open RESTful API specification [10].

2.1 ABAC Components

The ReCRED ABAC architecture includes three main functional components: the User, the Issuer and the Verifier. A description of these components from the functional point of view is provided in Figure 1 and in the following subsections, while a description of the actual mapping of these to the ReCRED general architectural entities is included in Section 2.2.

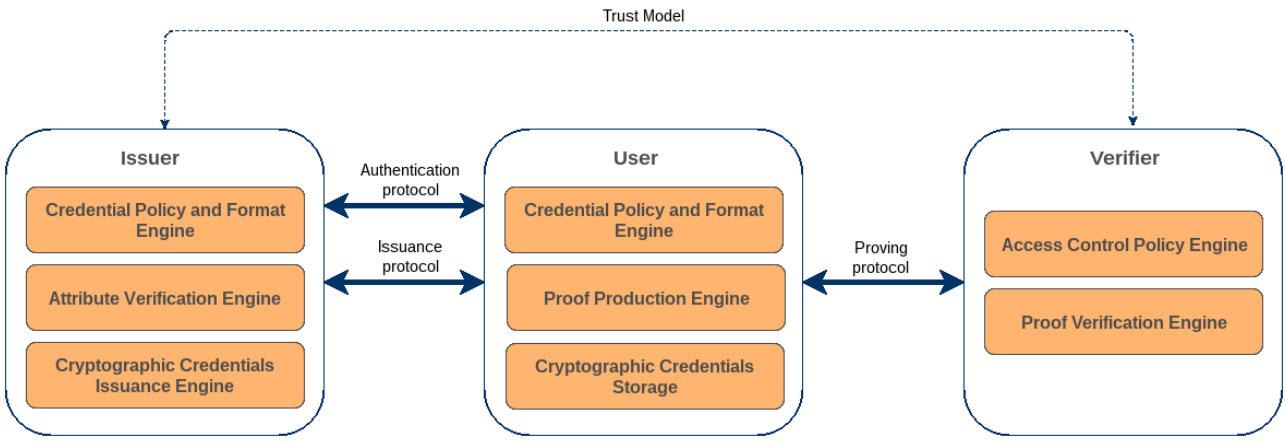

Figure 1 ReCRED ABAC functional architecture

2.1.1 User

In the ReCRED ABAC architecture, the User is the subject that wants to access a resource. It owns verifiable attributes and can hold and receive credentials from the Issuer. It can securely authenticate to Issuers and correctly parse credential policies. It can also send to the Verifier certified proofs (which can be zero-knowledge proofs) of possession of its attributes.

These functionalities are supported by:

- the User Credential Policy and Format Engine, which can correctly interpret the semantics of the credential policies and specifications provided by the Issuer
- the Proof Production Engine, which can output partial verifiable profiles including only a subset of the User attributes or a set of assertions based on them
- the Cryptographic Credentials Storage, which can securely store the cryptographic credentials obtained by the issuer

The User takes part both in the Issuing and Authentication protocols together with the Issuer and in the Proving protocol with the Verifier.

2.1.2 Issuer

In the ReCRED ABAC architecture, the Issuer is the entity that is able to verify the attributes of the User and to issue credentials that certify these attributes.

These functionalities are supported by:

- the Issuer Credential Policy and Format Engine, which can associate policies and credential types with the issuance action initiated by the User
- the Attribute Verification Engine, which can verify the attributes owned by the credential requesting User
- the Cryptographic Credential Issuance Engine, which can produce cryptographic credentials based on the verified attributes of the User

The Issuer takes part in the Authentication and Issuing protocols together with the User.

2.1.3 Verifier

In the ReCRED ABAC architecture, the Verifier is the entity that holds a resource that the User wants to access. It can verify the proofs provided by the User according to given policies, provided that a chain of trust exists towards the credential issuer.

These functionalities are supported by:

- the Access Control Policy Engine, which provides the attribute-based policies associated to requested resources
- the Proof Verification Engine, which can verify, according to a given policy, the partial verifiable profiles provided by the User

The Verifier takes part in the Proving protocol together with the User.

2.2 ReCRED Components Mapping to the ABAC Architecture

This section describes how the ABAC components described above (User, Issuer and Verifier) are mapped to the ReCRED Reference Architecture described in Deliverable D2.3. To this aim, Figure 2 shows the ReCRED architecture from an ABAC perspective.

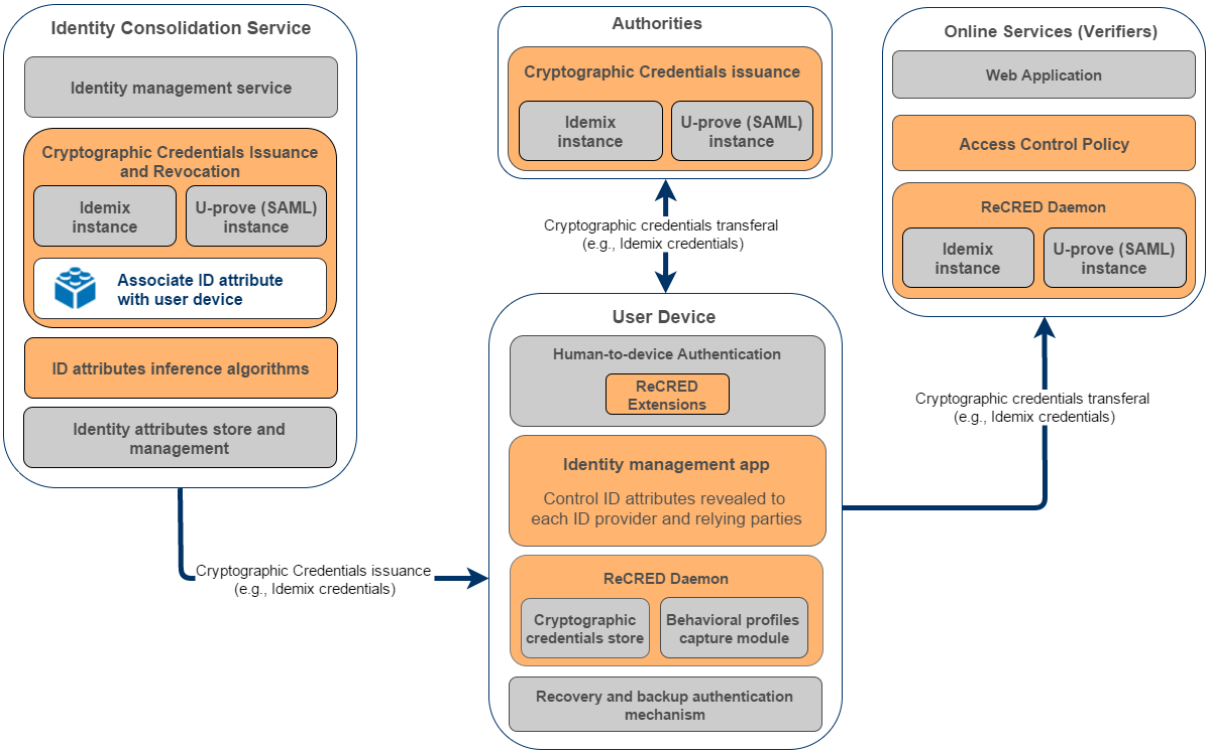

Figure 2 ABAC component view of the ReCRED Architecture

2.2.1 User Device

The User Device is the central component of the ReCRED architecture. From the ABAC point of view it maps to the User functionality described in Section 2.1.1. It has the capability to securely authenticate to Issuers and the Identity Consolidator (e.g. through FIDO [11], OpenID [12], OpenAuth [13]), to request cryptographic credentials from Issuers and the Identity Consolidator, to securely store cryptographic credentials, to backup credentials in the Identity Consolidator, to correctly parse and show Issuer and Verifier policies to the user and to create verifiable partial profiles containing a subset of identity attributes.

The Cryptographic Credentials Storage functionality is provided in the ReCRED architecture by the component with the same name, while the Proof Production Engine and the User Credential Policy and format engine functionalities are provided by the Identity Management application.

2.2.2 Identity Consolidator

The Identity Consolidator is the ReCRED component that enables horizontal identity binding and vertical real-to-online identity mapping. It can acquire the physical attributes of real world identities. From the ABAC point of view it maps to the Issuer functionality described in Section 2.1.2. It has the capability to allow secure authentication from the User Device, to issue verified cryptographic credentials to the user device, to allow backup credential storage from the user device, to prove identity attributes on behalf of the user and to receive identity attributes from the Identity Providers.

The Attribute Verification Engine, Issuer Credential Policy and Format Engine and Cryptographic Credential Issuance Engine functionalities are provided by the Cryptographic Credentials Issuance and Revocation module.

2.2.3 Identity Providers/Authorities

In the ReCRED architecture the Identity Providers are the entities that are able to verify (offline) the identity of the users. From the ABAC point of view they map to the Issuer functionality described in Section 2.1.2. They are able to allow secure authentication from the User Device, to issue credentials to the User Device, to transfer issued credentials as a backup to the Identity Consolidator on behalf of the user and, when privacy-aware cryptographic protocols are not supported, to transfer identity attributes to the Identity Consolidator so that this can issue credentials on behalf of the Identity Provider.

The Attribute Verification Engine, Issuer Credential Policy and Format Engine and Cryptographic Credential Issuance Engine functionalities are provided by the Cryptographic Credentials Issuance module.

2.2.4 Online Services (Verifiers)

The Online Services (Verifiers) are the entities of the ReCRED architecture that hold the resources that the user wants to access through her device.

From the ABAC point of view they map to the Verifier functionality described in Section 2.1.3. They have the capability to provide complex access-right policies to the users, to verify cryptographic credentials against these policies and to accordingly grant or deny access to the resources.

The Access Control Policy Engine and Proof Verification Engine functionalities are provided by the Access Control Policy and ReCRED Daemon modules.

2.3 Common Interfaces

The Idemix and U-Prove prototypes described in the sections 2.4.2 and 2.5.5 below employ common protocols and interfaces that are described in this section. Later in the project, as explained in sections 2.6 and 2.8 these will be extended to allow ABE-based access control as well.

2.3.1 Protocols

2.3.1.1 Authentication Protocol

The Authentication Protocol allows the Issuer to authenticate the User requesting a credential. This authentication can occur through FIDO UAF (which allows online services to offer user authentication based on a password-less mechanism), OpenID or OpenAuth.

2.3.1.2 Issuance Protocol

The Issuance Protocol runs between the User and the Issuer. It is initiated by the User requesting to the Issuer the policy for credential issuance (e.g. the Idemix credential structure). This is sent by the Issuer back to the User. Then the User authenticates to the Identity Provider through the Authentication Protocol. The User provides, according to the issuance policy, its attributes, which are then verified by the Issuer. If the verification is successful, the Issuer produces a cryptographic credential based on these attributes and sends it to the User.

2.3.1.3 Proving Protocol

The Proving Protocol runs between the User and the Verifier that holds a resource that the user wants to access. The protocol is initiated by the User requesting the attribute-based policy to access a given resource. The Verifier provides this policy. Based on it, the User is able to create a proof of possession for a selected subset of its attributes. This proof is sent to the Verifier who verifies it against its policies leveraging on its trust in the credential Issuer. If the verification is successful, the Verifier provides access to the requested resource to the User. Note that no authentication is needed between the User and the Online Service.

2.3.2 FIWARE

FIWARE is an open platform that aims to provide a novel service infrastructure to an easier development of future Internet applications. The FIWARE project is the core part of the Future Internet PPP program, a joint initiative by the European Industry and the European Commission.

The open and independent FIWARE community is formed by many individuals and organizations that agree on the FIWARE mission: “to build an open sustainable ecosystem around public, royalty-free and implementation-driven software platform standards that will ease the development of new Smart Applications in multiple sectors”.

The FIWARE platform is the heart of the project and it is mainly focused on technological aspects. Nevertheless, other non-technical relevant activities like FIWARE Lab, FIWARE Accelerator, FIWARE mundus or FIWARE iHubs are important components of the FIWARE ecosystem.

The novel FIWARE infrastructure is built upon basic atomic elements called **generic enablers (GE)**. These are software tools that provide a number of general-purpose functions and are freely available in the rich library of the FIWARE Catalogue. This design makes the development of new services quicker and easier by combining more GE in a modular approach.

Furthermore, each GE is offered through well-defined API (Application Programming Interfaces), easing the development of smart applications in multiple sectors. The set of FIWARE **RESTful API** specifications [10] cover a wide range of different application domains like Cloud Hosting, Internet of Things, security or networks and devices interfaces.

Related to the security area, more security and privacy aspects are in turn taken into consideration and the APIs for Privacy-preserving Authentication in an attribute based infrastructure are well defined too. FIWARE security specifications are based on the ABC4Trust specifications [8] which propose a cryptographic agnostic attribute credential protocol, thus supporting both Idemix and U-Prove. In particular, the FIWARE Privacy GE (Generic Enabler) specifies the API for a P2ABC (Privacy-Preserving Attribute Based Credentials) system.

These APIs describe the endpoints of the main roles that take place in an anonymous authentication system, i.e. Issuers, Users and Verifiers. The APIs are RESTful, resource-oriented, and are accessed via HTTP using XML-based representations for information interchange.

In Table 1, Table 2 and Table 3 the FIWARE APIs for Issuer, User and Verifier are summarized. The tables specify the HTTP method, the URL path, the data given as input inside the body of the request and the returned data in the response. If some parameters can be specified along with the URL path, these are listed above the path.

Table 1 Issuer API

Method	Path	input	output	description
GET	/issuer/setupSystemParameters/ • SecurityLevel • cryptoMechanism		SystemParameters	generates a fresh set of system parameters for the given security level
POST	/issuer/setupIssuerParameters/	IssuerParametersInput	IssuerParameters	generates a fresh issuance key and the corresponding issuer parameters
POST	/issuer/initIssuanceProtocol/	IssuancePolicyAndAttributes	IssuanceMessageAndBoolean	Initiate an issuance protocol based on the given issuance policy and the list of attributes to be embedded in the new credential.
POST	/issuer/issuanceProtocolStep/	IssuanceMessage	IssuanceMessageAndBoolean	performs one step in an interactive issuance protocol
GET	/issuer/getIssuanceLogEntry/ • issuanceEntryUid		IssuanceLogEntry	looks up an issuance log entry of previously issued credentials

Table 2 User API

Method	Path	input	output	description
POST	/user/canBeSatisfied/	PresentationPolicyAlternatives	ABCEBoolean	on input a presentation policy, decides whether the credentials in the user's credential store could be used to produce a valid presentation token satisfying the policy

POST	/user/createPresentationToken/	PresentationPolicyAlternatives	UiPresentationArguments	returns an argument to be passed to the UI for choosing how to satisfy the policy
POST	/user/createPresentationTokenUi/	UiPresentationReturn	PresentationToken	generates a presentation token that reflects this choice, and which satisfies the respective presentation policy alternatives
POST	/user/issuanceProtocolStep	IssuanceMessage	IssuanceReturn	performs one step in an interactive issuance protocol
POST	/user/issuanceProtocolStepUi/	UiIssuanceReturn	IssuanceMessage	Called after the user has made her choice on how to satisfy the issuance policy
POST	/user/updateNonRevocationEvidence/			updates the non-revocation evidence associated to all credentials in the credential store
GET	/user/listCredentials/		URISet	This method returns an array of all unique credential identifiers (UIDs) available in the Credential Manager
GET	/user/getCredentialDescription/{credentialUid} • credentialUid		CredentialDescription	returns the description of the credential with the given unique identifier.
DELETE	/user/deleteCredential/ • credUid		ABCEBoolean	deletes the credential with the given identifier from the credential store

Table 3 verifier API

Method	Path	input	output	description
POST	/verification/verifyTokenAgainstPolicy/ • **store**	PresentationPolicyAlternativesAndPresentationToken	PresentationTokenDescription	Checks whether the token satisfies the policy and checks the validity of the cryptographic evidence included in token. Stores the token in a dedicated store if *store* is set to true
GET	/verification/getToken/ • **tokenUID**		PresentationToken	looks up a previously verified presentation token

POST	/verification/deleteToken/ • **tokenUID**		Boolean	deletes the previously verified presentation token

2.3.2.1 Data format

The relevant data defined in FIWARE and involved in the system setup, issuance and verification phases are summarized above. Moreover, FIWARE specifies an *identity selection* component, involved both in the issuance and verification phases when the user has to choose a preferred combination of credentials and/or pseudonyms.

A more detailed description of the artifacts can be found at [10].

2.3.2.2 System parameters

The generic ABCE (ABC Engine) provides 4 setup methods:

- `setupSystemParameters`. This method generates new system parameters (e.g. size of secrets, size of moduli, prime probability, etc.)
- `setupIssuerParameters`. This method generates a secret issuance key and public issuer parameters.
- `setupRevocationAuthorityParameters`. This method generates a secret Revocation Authority key.
- `setupInspectionPublicKey`. This method generates a secret decryption key and a public encryption key for the Inspector.

The credential specification describes the contents of the credentials. It has the following form:

```
<abc:CredentialSpecification Version=”1.0” KeyBinding=”xs:boolean”
Revocable="xs:boolean">
 <abc:SpecificationUID>xs:anyURI</abc:SpecificationUID>
 <abc:numericalId>xs:integer</abc:numericalId>?
 <abc:FriendlyCredentialName xml:lang=”xs:language”/>*
 <abc:DefaultImageReference>xs:anyURI</abc:DefaultImageReference>?
 <abc:AttributeDescriptions MaxLength=”xs:unsignedInt”>
  <abc:AttributeDescription Type=”xs:anyURI” DataType=”xs:anyURI”
Encoding=”xs:anyURI”>
   <abc:FriendlyAttributeName
lang=”xs:language”>xs:string</abc:FriendlyAttributeName>*
   <abc:AllowedValue>...</abc:AllowedValue>*
  </abc:AttributeDescription>*
 </abc:AttributeDescriptions>
<abc:CredentialSpecification>
```

The elements presented above describe the following attributes:

```
/abc:CredentialSpecification
```

This element contains the credential content.

```
/abc:CredentialSpecification/@Version
```

This element specifies the version of the credential content.

```
/abc:CredentialSpecification/@KeyBinding
```

This element specifies whether the credential content is bound with a secret ley.

`/abc:CredentialSpecification/@Revocable`
This element specifies whether the credential content is revocable or not.

`/abc:CredentialSpecification/SpecificationUID`
This element contains an identifier for the credential specification.

`/abc:CredentialSpecification/abc:NumericalId`
This element contains a numerical identifier for the credential specification.

`/abc:CredentialSpecification/abc:FriendlyCredentialName`
This element contains a friendly name for the credential.

`/abc:CredentialSpecification/abc:FriendlyCredentialName/@lang`
This attribute contains a localization for the credential.

`/abc:CredentialSpecification/abc:DefaultImageReference`
This element contains a reference to the default image for the issued credential.

`/abc:CredentialSpecification/abc:AttributeDescriptions`
This element contains the description of the issued attributes.

`/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AttributeDescription`
This element contains the description of one attribute.

`/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AttributeDescription/@MaxLength`
This attribute specifies the maximal length in bits of the integers to which attribute values are mapped using the encoding function

`/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AttributeDescription/@Type`
This attribute contains a unique identifier of an attribute type.

`/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AttributeDescription/@DataType`
This attribute contains the data type of the credential attribute.

`/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AttributeDescription/@Encoding`
This attribute specifies the method for mapping an attribute to an integer value.

`.../abc:AttributeDescriptions/abc:AttributeDescription/FriendlyAttributeName`
This element contains a friendly name for the attribute.

`.../abc:AttributeDescriptions/abc:AttributeDescription/FriendlyAttributeName/@lang`
This attribute specifies a language identifier for the attribute friendly name.

`/abc:CredentialSpecification/abc:AttributeDescriptions/abc:AllowedValue`
This element specifies a list of permitted values for the attribute.

In order for multiple issuers to agree on the cryptographic parameters to use throughout the system, all entities in the system must agree on one set of system parameters. These parameters have to be generated once, before any of the issuer parameters and other keys are generated. The system parameters have the following form:

```
<abc:SystemParameters Version=”1.0” SystemParametersUID=”xs:anyURI”>
      <abc:CryptoParams>...</abc:CryptoParams>
</abc:SystemParameters>
```

The elements described above have the following meaning:

```
/abc:SystemParameters
```

This element contains the system parameters.

```
/abc:SystemParameters/@Version
```

This attribute specifies the system parameters version.

```
/abc:SystemParameters/@SystemParametersUID
```

This attribute specifies a unique identifier for the system parameters.

```
/abc:SystemParameters/abc:CryptoParams
```

This element contains the specific cryptographic elements for the system.

In order to issue credentials, the Issuer must specify system parameters, and generate a key pair consisting of a secret issuing key and a public verification key. The issuer parameters artifact is described below.

```
<abc:IssuerParameters Version=”1.0”>
<abc:ParametersUID>xs:anyURI</abc:ParametersUID>
<abc:FriendlyIssuerDescription
lang=”xs:language”>xs:string</abc:FriendlyIssuerDescription>*
<abc:AlgorithmID>xs:anyURI</abc:AlgorithmID>
<abc:SystemParametersUID>xs:anyURI</abc:SystemParametersUID>
<abc:MaximalNumberOfAttributes>xs:int</abc:MaxNumberOfAttributes>
<abc:HashAlgorithm>xs:anyUID</abc:HashAlgorithm>
<abc:CryptoParams>...</abc:CryptoParams>
<abc:RevocationParametersUID>...</abc:RevocationParametersUID>?
</abc:IssuerParameters>
```

The issuer parameters elements presented above describe the following attributes:

```
/abc:IssuerParameters
```

This element contains the issuer public parameters.

```
/abc:IssuerParameters/@Version
```

This attribute contains the version for the issuer parameters.

```
/abc:IssuerParameters/abc:ParametersUID
```

This element contains an identifier for the issuer parameters.

```
/abc:IssuerParameters/abc:FriendlyIssuerDescription
```

This element contains a friendly description for the issuer parameters.

```
/abc:IssuerParameters/abc:FriendlyIssuerDescription/@lang
```

This attribute contains a localization for the issuer parameters friendly name.

```
/abc:IssuerParameters/abc:AlgorithmID
```

This element contains the algorithm for the issuer parameters. The algorithms are Idemix (urn:abc4trust:1.0:algorithm:idemix) and U-Prove (urn:abc4trust:1.0:algorithm:U-Prove).

```
/abc:IssuerParameters/abc:SystemParametersUID
```

This element contains an identifier for the system parameters used with the described issuer parameters.

`/abc:IssuerParameters/abc:MaximalNumberOfAttributes`
This element specifies the maximum number of issued attributes.

`/abc:IssuerParameters/abc:HashAlgorithm`
This element identifies the hash algorithm which will be used to generate the presentation token.

`/abc:IssuerParameters/abc:CryptoParams`
This element contains cryptographic element specific to the employed algorithm.

`/abc:IssuerParameters/abc:RevocationParametersUID`
This element contains an identifier for the revocation authority.

In order for the Verifier to communicate to the User which cryptographic algorithms it supports, and provide additional parameters for these algorithms, the verifier must generate a set of verifier parameters and send them to the User. How this artifact is protected (authenticated) is application specific.

```
<abc:VerifierParameters Version=”1.0” VerifierParametersId=”xs:anyURI”
SystemParametersId=”xs:anyURI”>
       <abc:CryptoParams>...</abc:CryptoParams>
</abc:VerifierParameters>
```

2.3.2.3 Issuance

ABC4Trust and FIWARE propose an identical set of RESTful protocol message specifications. Regarding the attribute token issuance, the user cannot choose or bias the value assigned to the attribute.

The issuer publishes or sends to the User an issuance policy consisting of a presentation policy (which credentials the user must possess in order to be issued an attribute token) and a credential template. The user prepares a special presentation token that fulfills the stated presentation policy, but that contains additional cryptographic information to enable carrying over attribute, user binding, and device binding information.

The User and Issuer subsequently engage in a multi-round issuance protocol, at the end of which the user obtains the requested credential.

ABC4Trust specifies 2 types of issuance: simple issuance (e.g. regular U-Prove issuance) and advanced issuance (the attributes are derived from an existing token). We will describe the simple issuance variant.

The issuance steps are as follows:

1. The user authenticates to the issuer
2. The user specifies which attributes will be issued.
3. The issuer will invoke a generic `initIssuanceProtocol()` method with a set of attributes that shall be certified in the new credential and with an issuance policy that only contains the identifiers of the credential specification and the issuer parameters of the credential that is to be issued. This call initiates an action in the (CE) Crypto Engine (an entity responsible with the underlying crypto implementation – U-Prove). This method returns an issuance message sent to the user.
4. Both sides call a generic `issuanceProtocolStep()` which is called until a credential is issued.

By using such a strategy the issuance protocol is implementation agnostic.

Any message exchanged in the issuance protocol will be wrapped as an issuanceMessage. Because the issuance protocol contains multiple steps, each message includes a Context attributes.

```
<abc:IssuanceMessage Context=”...”>
</abc:IssuanceMessage>
```

```
/abc:IssuanceMessage
```

This element contains an issuance policy, issuance token or a mechanism specific cryptographic data.

```
/abc:IssuanceMessage/@Context
```

This attribute contains a context for the issuance message.

On the server side, all issued tokens must be logged.

When the issuance protocol is completed, the user obtains a credential which has the type CredentialDescription.

```
<abc:CredentialDescription RevokedByIssuer=”xs:boolean”?>
       <abc:CredentialUID>
       ...
       </abc:CredentialUID>
       <abc:FriendlyCredentialName lang=”xs:language”>
              xs:string
       </abc:FriendlyCredentialName>*
       <abc:ImageReference>
              xs:anyURI
       </abc:ImageReference>?
       <abc:CredentialSpecificationUID>
              ...
       </abc:CredentialSpecificationUID>
       <abc:IssuerParametersUID>
              ...
       </abc:IssuerParametersUID>
       <abc:SecretReference>...</abc:SecretReference>?
       <abc:Attribute>
              <abc:AttributeUID>...</abc:AttributeUID>
              <abc:AttributeDescription @Type=”xs:anyURI” @DataType=”xs:anyURI”
@Encoding=”xs:anyURI”>
                     <abc:FriendlyAttributeName lang=”xs:language”>
                     xs:string
                     </abc:FriendlyAttributeName>*
                     <abc:AttributeValue>...</abc:AttributeValue>
              </abc:AttributeDescription>
       </abc:Attribute>*
</abc:CredentialDescription>
```

```
/abc:CredentialDescription
```

This element contains the description of an issued credential.

```
/abc:CredentialDescription/@RevokedByIssuer
```

This attribute contains a flag which states the revocation status for the credential.

```
/abc:CredentialDescription/abc:CredentialUID
```

This element contains a unique identifier for the credential.

```
/abc:CredentialDescription/abc:FriendlyCredentialName
```

This element contains a friendly name for the credential.

`/abc:CredentialDescription/abc:FriendlyCredentialName/@lang`
This attribute contains the localization for the friendly name of the credential.

`/abc:CredentialDescription/abc:ImageReference`
This element contains a reference to the credential image location.

`/abc:CredentialDescription/abc:CredentialSpecificationUID`
This element contains an identifier for the credential specification.

`/abc:CredentialDescription/abc:IssuerParametersUID`
This element contains a reference to the issuer parameters.

`/abc:CredentialDescription/abc:SecretReference`
This element contains a local identifier for the secret key which is linked with the credential.

`/abc:CredentialDescription/abc:Attribute`
This element contains the description of an attribute.

`/abc:CredentialDescription/abc:Attribute/AttributeUID`
This element contains a local identifier for the attribute.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeDescription`
This element contains the description of an attribute.

`.../abc:Attribute/abc:AttributeDescription/@Type`
This attribute contains a unique identifier for the attribute type.

`.../abc:Attribute/abc:AttributeDescription/@DataType`
This attribute contains the data type of the attribute.

`.../abc:Attribute/abc:AttributeDescription/@Encoding`
This attribute specifies the encoding details of the attribute.

`/abc:CredentialDescription/abc:Attribute/abc:FriendlyAttributeName`
This element contains a friendly name for the attribute.

`.../abc:Attribute/abc:FriendlyAttributeName/@lang`
This attribute contains a localization for the attribute friendly name.

`/abc:CredentialDescription/abc:Attribute/abc:AttributeValue`
This element contains the actual attribute value.

FIWARE specifies that the issuance protocol messages must be wrapped by the Application into a security layer (FIWARE specification mention WS-Trust 1.4 [42]).

The Issuer is responsible for creating the Issuer Parameters and Credential Specifications. FIWARE specifies that the user must use at least one identity source and at least one attribute source (LDAP or JDBC compatible databases).

2.3.2.4 Token presentation

To provide certified information to a Verifier entity, the user must present a token which contain a series of attributes or statements regarding her credentials. Beside from revealing information about user credentials, the token can be used to sign messages, in order to ensure freshness.

The token must support paradigms like: pseudonyms, key binding, inspection and revocation. The Verifier can cryptographically verify the authenticity of a received presentation token using the credential specifications and issuer parameters of all credentials involved in the token. The Verifier must obtain the credential specifications and issuer parameters in a trusted manner, e.g., by using a traditional PKI to authenticate them or retrieving them from a trusted location. The process of presentation is triggered when the application on the user's side contacts a verifier to request access to a resource.

The presentation steps are as follows:

1. Presentation is triggered when user wants to access a protected resource
2. The Verifier responds with one or more presentation policies. It may specify which credentials from which trusted issuer are required, which attributes must be revealed etc.
3. The user invokes a generic method `createPresentationToken()`. The token is sent to the verifier.
4. The verifier calls a generic method `verifyTokenAgainstPolicy()`. This method verifies the token statements according to the presentation policy. If the verification succeeds, the token is stored.

The presentation policy sent by the Verifier has the following schema:

```
<abc:PresentationPolicyAlternatives Version=”1.0”>
<abc:PresentationPolicy PolicyUID=”xs:anyURI”?>
       <abc:Message>
              <abc:Nonce>...</abc:Nonce>?
              <abc:FriendlyPolicyName lang=”xs:language”>
                     xs:string
              </abc:FriendlyPolicyName>*
              <abc:FriendlyPolicyDescription lang=”xs:language”>
                     xs:string
              </abc:FriendlyPolicyDescription>*
              <abc:VerifierIdentity>
                     xs:any
              </abc:VerifierIdentity>?
              <abc:ApplicationData>
                     ...
              </abc:ApplicationData>?
       </abc:Message>?
<abc:Pseudonym Exclusive=”xs:boolean”? Scope=”xs:string”
Established=”xs:boolean”? Alias=”xs:anyURI”?SameKeyBindingAs=”xs:anyURI”?>
       <abc:PseudonymValue> </abc:PseudonymValue>?
</abc:Pseudonym>*
<abc:Credential Alias=”xs:anyURI”? SameKeyBindingAs=”xs:anyURI”?>
       <abc:CredentialSpecAlternatives>
              <abc:CredentialSpecUID>...</abc:CredentialSpecUID>+
       </abc:CredentialSpecAlternatives>
       <abc:IssuerAlternatives>
              <abc:IssuerParametersUID RevocationInformationUID=”xs:anyURI”?>
                     ...
              </abc:IssuerParametersUID>+
       </abc:IssuerAlternatives>
       <abc:DisclosedAttribute AttributeType=”xs:anyURI”
DataHandlingPolicy=”xs:anyURI”?>
              (      <abc:InspectorAlternatives>
                            <abc:InspectorPublicKeyUID>
                                   ...
                            </abc:InspectorPublicKeyUID>+
                     </abc:InspectorAlternatives>
```

```
                    <abc:InspectionGrounds>
                           ...
                    </abc:InspectionGrounds>
             )?
      </abc:DisclosedAttribute>*
</abc:Credential>*
<abc:VerifierDrivenRevocation>
      <abc:RevocationParametersUID>...</abc:RevocationParametersUID>
      <abc:AttributeCredentialAlias=”xs:anyURI” AttributeType=”xs:anyURI”>+
</abc:VerifierDrivenRevocation>*
<abc:AttributePredicate Function=”xs:anyURI”>
      (     <abc:Attribute CredentialAlias=”xs:anyURI” AttributeType=”xs:anyURI”
DataHandlingPolicy=”xs:anyURI”?/>
             |
             <abc:ConstantValue>...</abc:ConstantValue>
      )+
</abc:AttributePredicate>*
</abc:PresentationPolicy>+
</abc:PresentationPolicyAlternatives>
```

The presentation of one or multiple credentials results in a presentation token that is sent to the verifier. The syntax for the element is:

```
<abc:PresentationToken Version=”1.0”>
      <abc:PresentationTokenDescription PolicyUID=”xs:anyURI”
TokenUID=”xs:anyURI”?>
             <abc:Message>
                    <abc:Nonce>...</abc:Nonce>?
                    <abc:FriendlyPolicyName lang=”xs:language”>
                           xs:string
                    </abc:FriendlyPolicyName>*
                    <abc:FriendlyPolicyDescription lang=”xs:language”>
                           xs:string
                    </abc:FriendlyPolicyDescription>*
                    <abc:VerifierIdentity>xs:any</abc:VerifierIdentity>
                    <abc:ApplicationData>...</abc:ApplicationData>?
             </abc:Message>?
             <abc:Pseudonym Scope=”xs:string”? Exclusive=”xs:boolean”?
Alias=”xs:anyURI”? SameKeyBindingAs=”xs:anyURI”?>
                    <abc:PseudonymValue>...</abc:PseudonymValue>
             </abc:Pseudonym>*
             <abc:Credential Alias=”xs:anyURI”? SameKeyBindingAs=”xs:anyURI”?>
                    <abc:CredentialSpecUID>...</abc:CredentialSpecUID>
                    <abc:IssuerParametersUID>...</abc:IssuerParametersUID>
                    <abc:RevocationInformationUID>
                           ...
                    </abc:RevocationInformationUID>?
                    <abc:DisclosedAttribute AttributeType=”xs:anyURI”
DataHandlingPolicy=”xs:anyURI”?>
                           (
      <abc:InspectorPublicKeyUID>...</abc:InspectorPublicKeyUID>
                                  <abc:InspectionGrounds>...</abc:InspectionGrounds>
                           )?
                           <abc:AttributeValue>...</abc:AttributeValue>
                    </abc:DisclosedAttribute>
             </abc:Credential>*
             <abc:VerifierDrivenRevocation>

      <abc:RevocationInformationUID>...</abc:RevocationInformationUID>
                    <abc:Attribute AttributeType=”xs:anyURI” CredentialAlias=”
xs:anyURI” >+
                    </abc:VerifierDrivenRevocation>*
                    <abc:AttributePredicate Function=”xs:anyURI”>
                           (     <abc:Attribute CredentialAlias=”xs:anyURI”
                    AttributeType=”xs:anyURI” DataHandlingPolicy=”xs:anyURI”?/>
```

```
        |
        <abc:ConstantValue>...</abc:ConstantValue>
                             )+
        </abc:AttributePredicate>*

</abc:PresentationTokenDescription>
<abc:CryptoEvidence>
...
</abc:CryptoEvidence>
</abc:PresentationToken>
```

`/abc:PresentationToken`
This element contains a presentation token.

`/abc:PresentationToken/@Version`
This attribute contains the presentation token version.

`/abc:PresentationTokenDescription`
This element contains the description for the disclosed attributes.

`.../abc:PresentationPolicy/@PolicyUID`
This attribute contains an identifier for the presentation policy.

`.../abc:PresentationPolicy/@TokenUID`
This attribute contains a unique identifier for the presentation token.

`.../abc:PresentationTokenDescription/abc:Message`
This element contains a signed message by private key of each cryptographic token (credential).

`.../abc:PresentationTokenDescription/abc:Message/abc:Nonce`
This element contains a random number signed by the private key of each token.

`.../abc:PresentationTokenDescription/abc:Message/abc:FriendlyPolicyName`
This element contains a friendly name for the description.

`.../abc:PresentationTokenDescription/abc:Message/abc:FriendlyPolicyName/@lang`
This attribute specifies the localization for the friendly name policy.

`.../abc:PresentationTokenDescription/abc:Message/abc:VerifierIdentity`
This element contains the identity of the verifier.

`.../abc:PresentationTokenDescription/abc:Message/abc:FriendlyPolicyDescription`
This element contains a friendly description for the policy.

`.../abc:Message/abc:FriendlyPolicyDescription/@lang`
This attribute contains a localization for the friendly description of the policy.

`.../abc:PresentationTokenDescription/abc:Message/abc:ApplicationData`
This element contains the data type of the string.

`.../abc:PresentationTokenDescription/abc:Pseudonym`
This element indicates that a pseudonym is present along with the presentation token.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@Scope`
This attribute contains the scope of the pseudonym.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@Exclusive`
This attribute indicates that the pseudonym is scope-exclusive.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@Alias`
This attribute contains an alias for the pseudonym.

`.../abc:PresentationTokenDescription/abc:Pseudonym/@SameKeyBindingAs`
This attribute contains an alias to another pseudonym or to a Credential element for a credential key binding.

`.../abc:PresentationTokenDescription/abc:Pseudonym/abc:PseudonymValue`
This element contains the base64 encoding of the pseudonym.

`.../abc:PresentationTokenDescription/abc:Credential`
This element contains the presentation token credential.

`.../abc:PresentationTokenDescription/abc:Credential/@Alias`
This attribute contains an alias for the credential.

`.../abc:PresentationTokenDescription/abc:Credential/@SameKeyBindingAs`
This attribute contains an alias for a pseudonym or a reference to another Credential element for credential with key binding.

`.../abc:Credential/abc:CredentialSpecUID`
This element contains an identifier for the credential.

`.../abc:PresentationTokenDescriptionabc:Credential/abc:IssuerParametersUID`
This element contains an identifier for the credential public key.

`.../abc:PresentationTokenDescriptionabc:Credential/abc:RevocationInformationUID`
This element contains an identifier for the revocation information.

`.../abc:PresentationTokenDescription/abc:Credential/abc:Attributes`
This element contains the disclosed attributes.

`.../abc:PresentationTokenDescription/abc:Credential/abc:DisclosedAttribute`
This element contains one disclosed attribute.

`.../abc:Credential/abc:DisclosedAttribute/@AttributeType`
This element describes the credential type of the disclosed attribute.

`.../abc:Credential/abc:DisclosedAttribute/@DataHandlingPolicy`
This attribute contains an external data handler policy.

`.../abc:Credential/abc:DisclosedAttribute/abc:InspectorPublicKeyUID`
This optional element contains the identifier of the inspector public key under which the attribute value is encrypted

`.../abc:Credential/abc:DisclosedAttribute/abc:InspectionGrounds`
This element contains the context under which the inspector can decrypt the attributes.

`.../abc:Credential/abc:DisclosedAttribute/abc:AttributeValue`
This element contains the base64 encoding of the disclosed attribute.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation`
The element describes the parameters used in the validity verification of an attribute.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:RevocationInformationUID`
This element contains an identifier for the verification information.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:Attribute`
This element specifies a credential attribute that is used for verifier-driven revocation

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:Attribute/@CredentialAlias`
This attribute describes an alias for the credential from which the attribute was used.

`.../abc:PresentationTokenDescription/abc:VerifierDrivenRevocation/abc:Attribute/@AttributeType`
This attribute refers to the attribute used for verifier-driven information.

`.../abc:PresentationTokenDescription/abc:AttributePredicate`
This optional element specifies a predicate that is guaranteed to hold by this token.

`.../abc:AttributePredicate/@Function`
This attribute specifies the boolean function for this predicate.

`.../abc:AttributePredicate/abc:Attribute`
This element contains a reference to the attribute used in the predicate evaluation.

`.../abc:AttributePredicate/abc:Attribute/@CredentialAlias`
This attribute contains an alias for the credential which contains the attribute.

`.../abc:AttributePredicate/abc:Attribute/@AttributeType`
This attribute contains the exact attribute used as a predicate argument.

`.../abc:AttributePredicate/abc:Attribute/@DataHandlingPolicy`
This attribute contains an external data handling policy used in predicate evaluation.

`.../abc:AttributePredicate/abc:ConstantValue`
This element specifies a constant value used for the predicate.

`/abc:PresentationToken/abc:CryptoEvidence`
This element contains the cryptographic elements for the presentation token.

2.3.2.5 Identity Selection

The Identity Selection steps are executed between the user and the identity selection component. These allow the user to choose among different combination of credentials and/or pseudonyms in order to satisfy a presentation or an issuance policy. The identity selection component can be considered as user interface (UI), for instance a graphic interface. More details can be found in the FIWARE specification [9].

2.3.2.6 Credential revocation

The revocation process is executed by a Revocation Authority which generates and publishes its parameter at setup. The Revocation Authority could be a separate entity or be represented by the Verifier or Issuer entities. In contrast with a classical public key authentication system, where revoked certificate serials are published in a CRL (Certificate Revocation List), a privacy preserving attribute system cannot employ such a mechanism because it would require a unique identification number for the user credential. By having a unique identification number the unlinkability and the untraceability requirements would not be satisfied.

In a revocation scheme there is a Revocation Authority which must provide to the Verifier the latest revocation information or the revocation information must be signed if this is presented along with the credentials in the presentation protocol.

The revocation system could require a non-revocation proof from the user side. In this case the user must deliver a non-revocation proof for each credential and Revocation Authority against which the latter credential is verified. The first non-revocation evidence is obtained by the user from the Revocation Authority the first time it checks the credential. The non-revocation evidence could be updated if the application scenario or if the revocation policy requires it. In order to speed up the presentation protocol the user could have a proactive approach and obtain in advance the non-revocation evidence.

Depending on the revoked information context there are 2 types of revocation: issuer-driven (global) and verifier-driven (context specific). Regarding the issuer-driven revocation the Issuer specifies the Revocation Authority against which the credentials must be verified. The issuer-driven revocation has a global scope and the verifier must validate the credential during the presentation protocol against the latest revocation information. Concerning the verifier-driven information the verifier specifies in the presentation protocol additional Revocation Authorities (beside the issuer-driven ones) against which the credentials must be checked. The revocation data is particular for the context of the Verified and does not impact the presentation protocols in which other verifiers are involved.

2.4 Idemix

Idemix [6][14][15] is an identity management system based on anonymous credentials and zero-knowledge protocols. Parties involved in the Idemix system can play the roles of issuers, recipients, provers and verifiers. The issuer represents the authority demanded to the issuance of credentials (for which she is responsible) to a recipient through an issuance protocol that results in the recipient owning a credential. When a proof of possession of credential is required by a verifier, the credential owner (the recipient that obtained the credential issued by the authority) acts in the role of a prover towards the verifier. An Idemix credential consists of a set of attribute values as well as cryptographic information that allows the owner of the credential to create the proof of possession. The components of the ReCRED reference architecture are easily mapped into the above described roles of the Idemix architecture, in order to realize an anonymous credential system to provide Attribute Based Access Control feature in the ReCRED framework. In this section we are going to present the protocols and the messages involved in the Idemix system described in [6] to which we refer for a complete description of the cryptographic system.

2.4.1.1 Idemix Anonymous Credential Scheme

The Idemix protocol requires that all parties agree on public master system parameters such as the bit length of all relevant parameters as well as the groups to be used. Given such global parameters, a user can choose her master secret key that will be contained by each credential, resulting in a parameter that binds together all credentials. This prevents from sharing credentials with the aim of collusion, as sharing one credential effectively implies sharing all the credentials of a user. Moreover, the master secret allows the prover to derive the pseudonyms to use when she requires credentials to be issued by the issuers. Each receiver can generate different pseudonyms to be shown to the issuers. Such pseudonyms, even if generated by the same receiver, cannot be linked to each other unless she proves that they are based on the same master secret key. Issuers generate public and secret keys associated to the cryptographic primitives they use and make the public keys available together with a specification of the services they offer. As an example, each issuer publishes the definition (i.e. the Credential Structure as shown in Section 2.4.2) of the credential it allows to be

issued. To obtain a credential, the receiver contacts an issuer and agrees with her on the structure of the credential, i.e. which will be the values of the attributes asserted by the credential. She then runs the interactive issuing protocol with the organization. Having acquired a credential, the receiver switches to the role of a prover in order to prove to a verifier the possession of the credential. A proof of possession may involve several credentials acquired by the same user or proving statements on the attribute values contained in the credentials using the proving protocol. Moreover, these proofs may be linked to a pseudonym chosen by the prover. Moreover, the proving protocol and issuing protocol credentials may be combined, in the case of an issuer requiring the recipient to release certified attribute values (a proof that she holds a credential issued by another party) before issuing a new credential. The Idemix protocol consists of three basic functionalities described in the following sections: i) *system setup*, allows parties to get initialized in the Idemix system, ii) *credential issuance*, is the functionality that permit a receiver to get the credential by the issuer and finally iii) *credential proving*, is the functionality demanded to the verification of credentials presented by a prover to a verifier.

2.4.1.2 System Setup

The anonymous credential system requires general parameters, which are separated into system parameters consisting of bit lengths and group parameters which define the groups that are used within the underlying cryptographic scheme.

- **Global Parameters:** System parameters should be fixed and made public to all parties. Such global parameters include bit lengths and groups size to be used in the scheme. Such parameter's values are reported in [6] appendix.
- **Issuer Parameters:** The issuer's key pair is used for issuing credentials to the users, i.e. issuing signatures on a lists of attributes requested by a user. The maximum number l of attributes of the credential is determined by the public key of the issuer. The number of attributes available to users is $l - \ell_{res}$ since some attributes, e.g. the master secret that is considered as an attribute, are reserved. The issuer generates:
  - *A safe RSA key-pair*: generates the safe primes p and q where $p = 2p' + 1$ and $q = 2q' + 1$, and then computes the RSA module $n = pq$
  - Random values $x_z, x_{R_1}, \dots, x_{R_l} \overset{R}{\leftarrow} \{2, p'q' - 1\}$ and $S \overset{R}{\leftarrow} QR_n$ in order to compute the CL signature [16][17] parameters:

$$Z = S^{x_z} \text{ and } R_i = S^{x_{R_i}}$$

  Finally, the issuer's public key is the set of parameters $pk = (n, S, Z, R_1, \dots, R_l, P)$, where P is the proof of correctness of the public key, while the private key is $sk = (p, q)$.

- **User Parameters**: The user's master secret m_1 is an integer chosen uniformly at random from the interval [1,ρ].
- **Pseudonyms:** The user can generate as many pseudonyms (*nym*) and domain pseudonyms (*dnym*) as she wants. Each pseudonym or domain pseudonym is un-linkable to any other pseudonym or domain pseudonym generated by the user. However, the domain pseudonyms enforce that a user can only generate one pseudonym per domain (i.e., given the domain and the user's master secret key, the domain pseudonym is unique).

2.4.1.3 Credential Issuance

The issuance phase is performed by running an interactive protocol between the Issuer and the Recipient (i.e. the User requesting the issuance of a credential owned and specified by the issuer she is contacting).

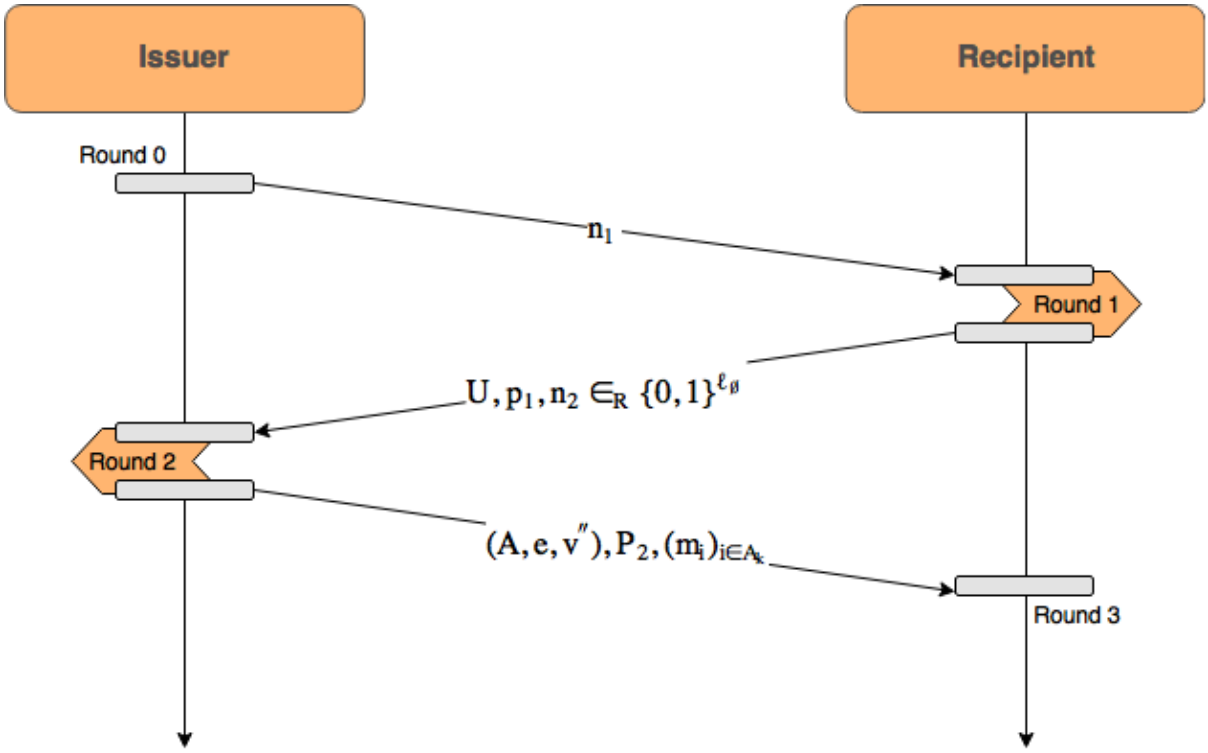

Figure 3 Idemix Issuance Protocol

As shown in Figure 3, the issuance protocol is the result of four different phases, namely rounds, performed by the two parties that exchange the output of each round in order to proceed with the following one:

- **Round 0:** Nonce Generation
  - **Performed by:** Issuer
  - **Input:** -
  - **Output:** n_1

The *Issuer* extract a random value n_1 to be provided to the *Recipient* and start the issuance protocol by using such a fresh value.

$$\boldsymbol{n_1} \xleftarrow{R} \{0,1\}^{\ell_0}$$

- **Round 1:** Attributes' commitment
  - **Performed by:** Recipient
  - **Input:** n_1
  - **Output:** U, P_1, n_2

The *Recipient* computes the value, considering the value of each attribute $\{m_i\}$:

$$\boldsymbol{U} = S^{v'} \cdot \prod_{j \in A} R_j^{m_j} \, mod \, n, \text{ where: } v' \overset{R}{\leftarrow} \pm \{0,1\}^{\ell_n + \ell_0}$$

This is the commitment of the attribute's values m_i to demonstrate to the issuer the ownership of such attributes by the recipient. As output of this round, the *Recipient* produces a non-interactive proof P_1 of the above computation:

$$\boldsymbol{P_1} = \begin{pmatrix} c = H(context | \, C_1 | \ldots | C_k | nym | dnym | \widetilde{U} | \tilde{C}_1 | \ldots | \tilde{C}_k | \widetilde{nym} | \widetilde{dnym} | n_1) \\ s_A = \{\hat{m}_0, \ldots, \hat{m}_k\} \, where \;\; \hat{m}_j = \tilde{m}_j + c m_j \\ \hat{v}' = \tilde{v}' + c v' \;\; where \\ \hat{r}_j = \tilde{r}_j + c r_j \;\; if \;\; nym \neq \perp \end{pmatrix}$$

The values included in P_1 are reported below:

1. Choose the random value:

$$\tilde{m}_j \overset{R}{\leftarrow} \{0,1\}^{\ell_0 + \ell_m + \ell_H + 1} \; for \; each \; attribute \; value$$

2. Proof of the knowledge of pseudonym and master secret m_1:

$$\widetilde{nym} = g^{\tilde{m}_1} h^{\tilde{r}_i} \;\; mod \;\; \Gamma, \;\; where \, \tilde{r}_i \overset{R}{\leftarrow} [0, \rho]$$

3. Proof of the knowledge of domain's pseudonym and master secret m_1:

$$\widetilde{dnym} = g_{dom}^{\tilde{m}_1} \;\; mod \;\; \Gamma$$

4. Knowledge of representation of U:

$$\widetilde{U} = S^{\tilde{v}'} \cdot \prod_{j \in A} R_j^{\tilde{m}_j} \, mod \, n \;\; \text{where } \tilde{v}' = \overset{R}{\leftarrow} \pm \{0,1\}^{\ell_n + 2\ell_0 + \ell_H}$$

5. Knowledge of committed values:

$$\tilde{C}_j = \{\tilde{C}_1, \dots, \tilde{C}_k\} \text{ with } \begin{cases} \tilde{C}_k = Z_k^{\tilde{m}_k} S_k^{\tilde{r}_k} \, mod \, n \\ \tilde{r}_j \overset{R}{\leftarrow} \{0,1\}^{2\ell_0+\ell_n+\ell_H} \end{cases}$$

6. Fiat-Shamir challenge:

$$c = H(context | \, C_1 | \dots | C_k | nym | dnym | \widetilde{U} | \tilde{C}_1 | \dots | \tilde{C}_k | \widetilde{nym} | \widetilde{dnym} | n_1)$$

7. Responses to the challenge:

$$\hat{v}' = \tilde{v}' + cv'$$

$$s_A = \{\hat{m}_0, \dots, \hat{m}_k\} \text{ with } \hat{m}_j = \tilde{m}_j + cm_j$$

In addition to the U and the proof of it P_1, the *Recipient* extract a random value n_2 to be used as a client generated *nonce* for the issuance session:

$$\boldsymbol{n_2} \overset{R}{\leftarrow} \pm \{0,1\}^{\ell_0}$$

- **Round 2:** Signature generation
  - **Performed by:** Issuer
  - **Input:** $U, p_1, n_2 \in_R \{0,1\}^{\ell_0}$
  - **Output:** $(A, e, v''), P_2, \{m_i\}_{i \in A_k}$

The *Issuer* verifies the correctness of the P_1 value sent by the *Recipient* by computing the following:

1. Knowledge of pseudonym and master secret m_1:

$$\widehat{nym} = nym^{-c} g^{\hat{m}_1} h^{\hat{r}} \; mod \; \Gamma$$

2. Knowledge of domain pseudonym and master secret m_1:

$$\widehat{dnym} = dnym^{-c} g_{dom}^{\hat{m}_1} \, mod \; \Gamma$$

3. Representation of U:

$$\hat{U} = U^{-c}(S^{\hat{v}'}) \cdot \prod_{j \in A} R_j^{\hat{m}_i} \, mod \, n$$

4. Knowledge of committed values:

$$\tilde{C}_j = \{\tilde{C}_1, \dots, \tilde{C}_k\} \text{ with } \hat{C}_k = c_k^{-c} Z_k^{\hat{m}_k} S_k^{\hat{r}_k} \, mod \, n$$

5. Challenge verification to be compared with c sent by *Recipient*:

$$\hat{c} = H(context | \, C_1 | \dots | C_k | nym | dnym | \hat{U} | \hat{C}_1 | \dots | \hat{C}_k | \widehat{nym} | \widehat{dnym} | n_1)$$

In case $c = \hat{c}$, the correctness is verified and the Issuer proceeds with the generation of the CL-signature σ_{CL} on attribute's values:

$$\boldsymbol{\sigma_{CL}} = \begin{pmatrix} \boldsymbol{A} = Q^{e^{-1}} \bmod p'q' \\ \boldsymbol{e} \overset{R}{\leftarrow} [2^{\ell_e - 1}, 2^{\ell_e - 1} + 2^{\ell'_e - 1}] \\ \boldsymbol{v''} = 2^{\ell_v - 1} + \tilde{v} \ \ with \ \ \tilde{v} \overset{R}{\leftarrow} \{0,1\}^{\ell_v - 1} \end{pmatrix}$$

1. The value Q is computed as follows:

$$Q = \frac{Z}{US^{v''} \prod_{j \in A_k} R_i^{m_i}} \ \ mod\ n$$

To proof the correctness of the computation, the *Issuer* compute the value P_2:

$$\boldsymbol{P_2} = \begin{pmatrix} c' = H(context|Q|A|\tilde{A}|n_2) \\ s_e = e - ce' \ \ mod\ p'q' \end{pmatrix}$$

1. Such computation also includes the definition of:

$$\tilde{A} = Q^r \ \ mod\ n \ \ \text{with}\ r \overset{R}{\leftarrow} \mathbb{Z}^*_{p'q'}$$

This phase of the issuance protocol ends with the *Issuer* sending to the *Recipient* the following values:

$$\sigma_{CL} = (A, e, v''), P_2, \{m_i\}_{i \in A_k}$$

- **Round 3:** Signature storage
  - **Performed by:** Recipient
  - **Input:** $(A, e, v''), P_2, \{m_i\}_{i \in A_k}$
  - **Output:** (A, e, v)

The last step of the issuance protocol, executed by the *Recipient*, starts with the verification of the CL-signature on attribute's values produced by the *Issuer*. Indeed, the *Recipient* first compute the value:

$$v = v'' + v'$$

Using that value checks whether:

$$Q = \frac{Z}{US^{v} \prod_{j \in A_k} R_i^{m_i}} \ \ mod\ n \ := \ \hat{Q} = A^e \ \ mod\ n$$

If $Q = \hat{Q}$ it proceed with the verification of the values in P_2:

1. Computes the value of $\hat{A}$:

$$\hat{A} = A^{c' + s_e\, e} S^{v' s_e} \ mod\ n$$

2. Checks whether:

$$\hat{c} = H(context|Q|A|\hat{A}|n_2) := c'$$

Finally stores the signature of attributes $\{m_i\}_{i\in A}$: $\boldsymbol{A}, \boldsymbol{e}, \boldsymbol{v}$

2.4.1.4 Credential Proving

The credential proving procedure, differently from the issuance protocol, is performed in only two steps, as shown in Figure 4. The credential proving phase aims to demonstrate to the verifier that a prover (i.e. the recipient of the issuance phase) effectively owns the combination of attributes that satisfy a given access control policy, namely *Statement S,* over such attributes. In the Idemix scheme, such policy is split in the so called *Predicates* so that for each predicate the corresponding Prover and Verifier algorithm exists. For ease of discussion we will present here only details for the verification of a set of attribute's values signature that is, de facto, the proof of possession of a set of attributes having specific values required by the verifier. The Idemix scheme allows also to prove and verify statement with more complex boolean policies between attributes, as described in [14].

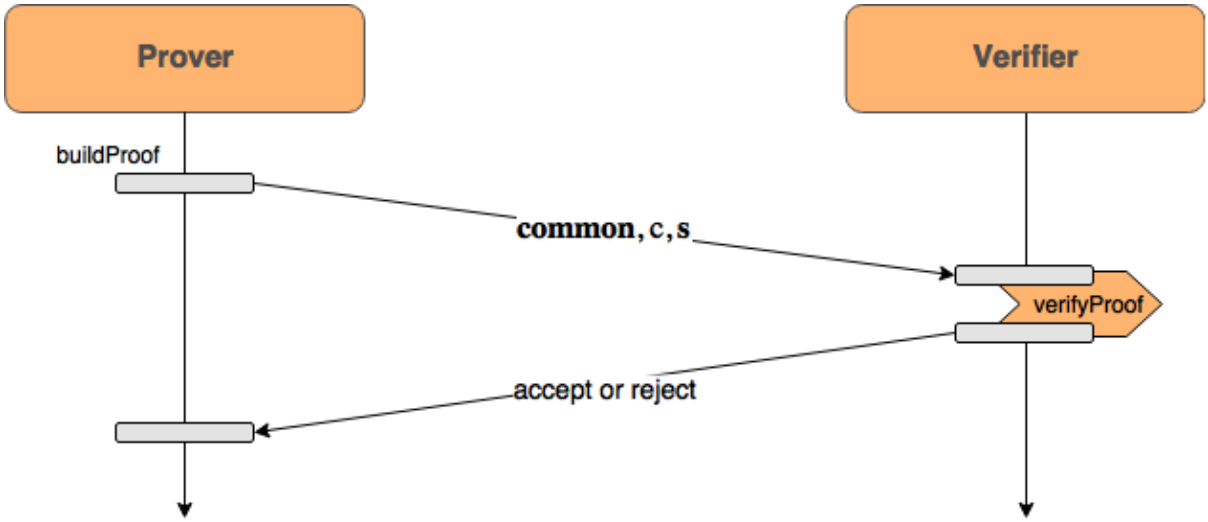

Figure 4 Idemix Proving Protocol

This phase is composed by a simple request-response protocol with the following two procedures:

- **buildProof:** performed by the prover (that was the recipient of the issuance phase) in order to produce the proof of possession of the credentials required by the verifier policy.
  - **Performed by:** Prover
  - **Input:** $m_1, \{cred\}, S, n_1$
  - **Output:** non-interactive proof of statements in S: (Common, c, s)

Where S is the statement sent by the verifier, $\{cred\}$ are the credential(s) and the nonce n_1 is generated by the *Verifier* and sent to the *Prover*. In what follows we show the building procedure for the proof of the signature (sub-prover ProveCL) for a set of attributes A_r *requested by* and *revealed to* the *Verifier.* The attributes not requested by the *Verifier* will not be revealed to preserve the user's privacy and reported as *hidden* attributes included in the set A_h. The prover maintains a global list of values that are common to all sub-provers algorithms of the build proof procedure:

1. Hidden value of each hidden attribute in A_h:

$$\widetilde{m}_i \overset{R}{\leftarrow} \{0,1\}^{\ell_0+\ell_m+\ell_H}$$

2. Common values **common** and t-values **T** (ProveCL in the example):
   1. Randomize the credential's signature (A, e, v):

$$r_A \overset{R}{\leftarrow} \{0,1\}^{\ell_0+\ell_n}$$

$$\sigma_R = \begin{pmatrix} \boldsymbol{A'} = AS^{r_A} \\ v' = v - er_A \\ e' = e - 2^{\ell_e-1} \end{pmatrix}$$

   2. Compute *t-values*:
      - $\tilde{e} \overset{R}{\leftarrow} \{0,1\}^{\ell_0+\ell_H+\ell'_e}$
      - $\tilde{v}' \overset{R}{\leftarrow} \{0,1\}^{\ell_0+\ell_H+\ell_v}$
      - For each attribute in A_h compute:

$$\tilde{Z} = (A')^{\tilde{e}} \left(\prod R_i^{\widetilde{m}_i}\right) \left(S^{\tilde{v}'}\right)$$

   At this point add A' to the common values list $\{\boldsymbol{common}\}$ and $\tilde{Z}$ to the *t-values* list $\{T\}$.

3. Compute the challenge:

$$\boldsymbol{c} = H(context|\{\boldsymbol{common}\}|\{T\}|n_1)$$

4. Compute the *s-values*:

$$\boldsymbol{s} = \begin{pmatrix} \hat{e} = \tilde{e} + c\left(e - 2^{\ell_e-1}\right) \\ \hat{v} = \tilde{v} + cv' \\ \hat{m}_i = \widetilde{m}_i + cm_i, \quad \forall m_i \in A_h \end{pmatrix}$$

The proof of possession of attributes A_r requested by the *Verifier* is: $(Common, c, s)$.

- **verifyProof**: is the verification phase of the protocol in which the verifier makes a decision on effective possession of required attributes by the prover based on the access policy (statement

S) and the proof message. In the case of the proof of possession of specific attributes with specific values as stated above, the verifier will run the *VerifyCL* sub-prover.

- **Performed by:** Verifier
- **Input:** $S, (Common, c, s), n_1$
- **Output:** accept or reject of the proof

1. Retrieve the common values and the s-values and compute:

$$\hat{T} = \left(\frac{Z}{(\prod_{i \in A_r} R_i^{m_i})(A')^{2^{\ell_e - 1}}}\right)^{-c} (A')^{\hat{e}} \left(\prod_{i \in A_h} R_i^{\hat{m}_i}\right)(S^{\hat{v}'})$$

2. Add $\hat{T}$ to the *t*-value verification list $\{\hat{T}\}$ and compute the challenge verification:

$$\hat{c} = H(context|\{\boldsymbol{common}\}|\{\hat{T}\}|n_1)$$

3. If the challenge verification $\hat{c}$ matches the challenge *c* sent by the *Prover*, the verification is successful, otherwise reject it.

2.4.2 Prototype

The aim of the Idemix prototype is the implementation of the Idemix instances that will be deployed at the User Device, Authorities, Identity Consolidator and Online Services (verifiers) architectural components. These will be integrated in the ReCRED daemon of the User Device, in the ReCRED daemon of the Online Services (verifiers) and in the Cryptographic Credentials Issuance (and Revocation) component of the Identity Consolidator and Authorities.

We strive at developing reusable and portable components that implement the basic Idemix ABAC functionalities as independent modules which can be easily integrated in heterogeneous applications. To this aim there are different Idemix prototype implementation efforts currently ongoing at the same time.

2.4.2.1 IBM Idemix library based prototype

Departing from the IBM Idemix library we have prototyped a set of RESTful services with a simplified API, which currently allows for a basic demonstration of an Idemix transaction. In this implementation the User is implemented as an HTTP client, while the Issuer and Verifier are implemented as Java-based Web services.

At the beginning of the issuance protocol, the User can request from the Issuer a credential policy that specifies which attributes can be issued, which is the credential structure and the public parameters of the Issuer. Having retrieved the issuance policy, the User should firstly authenticate herself and can then send to the Issuer her pseudonym along with her attributes. The session authentication protocol is out of the issuance one and depends on the specific use case. Finally, the Idemix multi round protocol can run and the credential can be retrieved and stored by the User.

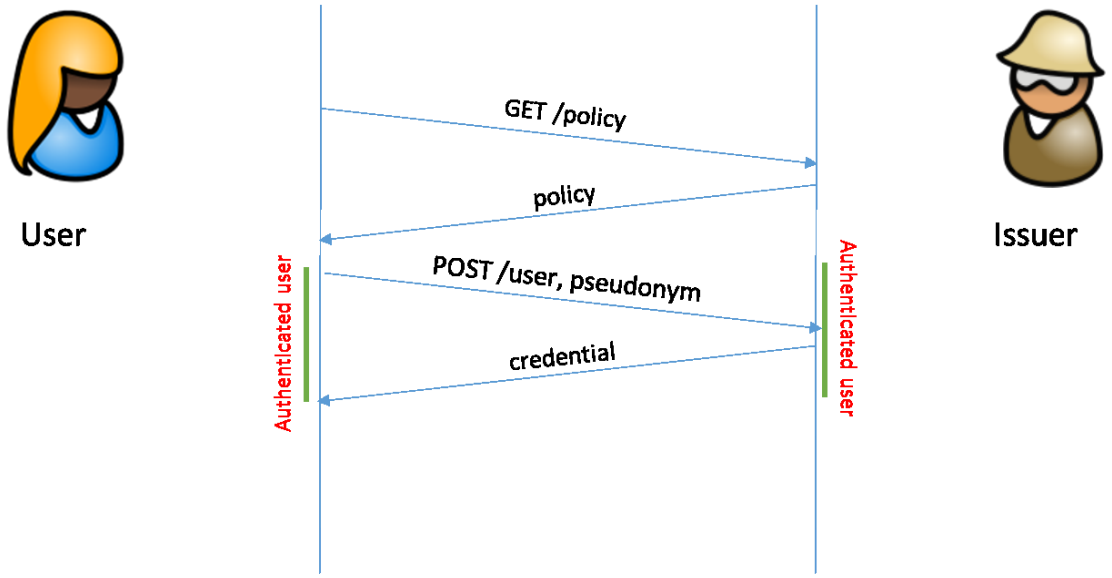

Figure 5 Simplified idemix issuance transaction

When the User want to access a resource, she firstly requests from the verifier the presentation policy that specifies the access criteria to be satisfied. Then, the User send her pseudonym to the Verifier and finally extracts a presentation token from her credential and send it to the Verifier. The Verifier is able to compare the presentation token against the presentation policy, allowing or denying the access to the resource.

Figure 5 and Figure 6 show the message flow of the simplified issuing and verification protocol. For each request/response pair we specified the method and the data exchanged. Furthermore, in Figure 5 we underlined the part of the transaction that requires a pre-authentication phase.

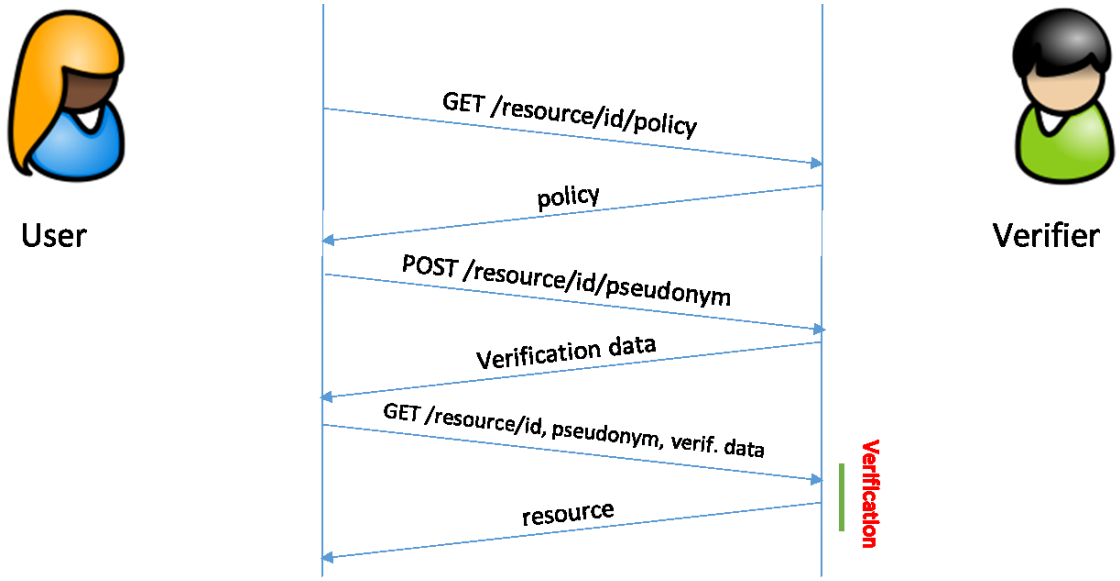

Figure 6 Simplified idemix verification transaction

2.4.2.2 FIWARE/ABC4Trust based prototype

While the implementation based on the IBM Idemix library described above is ongoing, a parallel prototype that aims at allowing a broader and standard interoperability is being developed. This prototype is based on the Open Privacy FIWARE generic enablers contributed by the ABC4Trust project [8], which are also described in Section 2.3.2 of this document.

The p2abcengine source code [21] provides a set of core components useful to setup an attribute based authentication ecosystem. The available components can be deployed in two ways: running independent web services that already expose RESTful interfaces, or integrating the code directly into a Java codebase. The first solution requires to compile the source code, then setup the various services and define the XML data structures. The integration solution can be carried out taking advantage of helper Java classes included in the p2abcengine core code. These classes encapsulate the most common setup for User, Issuer and Verifier.

The software design and implementation effort is ongoing towards a ReCRED application layer that uses the FIWARE API internally and also exposes this API externally as much as possible, taking into account the ReCRED requirements. An architectural overview of this implementation is provided in Figure 7.

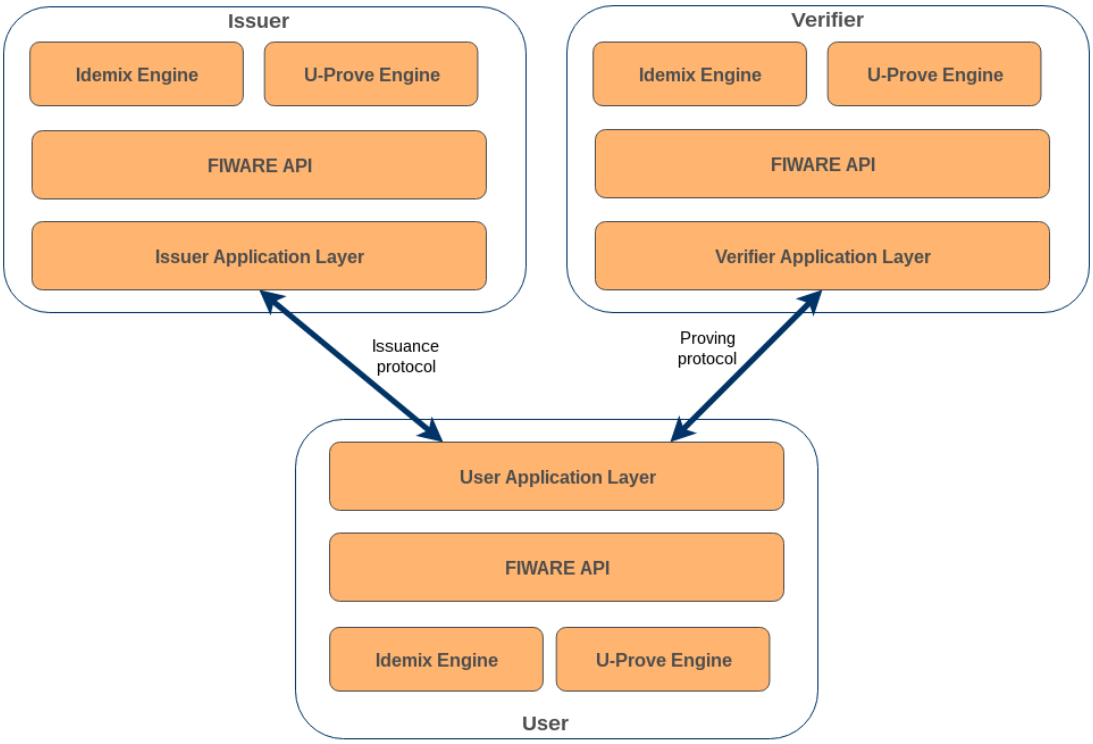

Figure 7 FIWARE based implementation general architecture

In this prototype, the Idemix parameters are serialized in the XML format defined by the FIWARE specification. As this specification is protocol agnostic, we here report only the sections that concern the Idemix implementation.

2.4.2.2.1 System Parameters

```
<abc:SystemParameters>
   <abc:CryptoParams>
       <abc:StringParameter Name="urn:idmx:3.0.0:systemParameters:hashFunction">
           <abc:Value>urn:abc4trust:1.0:hashalgorithm:sha-256</abc:Value>
       </abc:StringParameter>
       <abc:IntegerParameter
Name="urn:idmx:3.0.0:systemParameters:rsaModulusLength">
           <abc:Value>1024</abc:Value>
       </abc:IntegerParameter>
       <abc:IntegerParameter Name="urn:idmx:3.0.0:systemParameters:securityLevel">
           <abc:Value>73</abc:Value>
       </abc:IntegerParameter>
       <abc:StringParameter Name="urn:idmx:3.0.0:systemParameters:hashFunction">
           <abc:Value>urn:abc4trust:1.0:hashalgorithm:sha-256</abc:Value>
       </abc:StringParameter>
       <abc:IntegerParameter
Name="urn:idmx:3.0.0:systemParameters:rsaModulusLength">
           <abc:Value>1024</abc:Value>
       </abc:IntegerParameter>
       <abc:IntegerParameter
Name="urn:idmx:3.0.0:systemParameters:attributeLength">
           <abc:Value>256</abc:Value>
       </abc:IntegerParameter>
       <abc:IntegerParameter
Name="urn:idmx:3.0.0:systemParameters:dhModulusLength">
           <abc:Value>1024</abc:Value>
       </abc:IntegerParameter>
       <abc:IntegerParameter
Name="urn:idmx:3.0.0:systemParameters:dhSubgroupLength">
           <abc:Value>264</abc:Value>
       </abc:IntegerParameter>
       <abc:IntegerParameter Name="urn:idmx:3.0.0:systemParameters:statisticalZk">
           <abc:Value>80</abc:Value>
       </abc:IntegerParameter>
       <abc:IntegerParameter Name="urn:idmx:3.0.0:systemParameters:primeProb">
           <abc:Value>80</abc:Value>
       </abc:IntegerParameter>
       <abc:BigIntegerParameter Name="urn:idmx:3.0.0:systemParameters:dhModulus">
           <abc:Value>29...9634</abc:Value>
       </abc:BigIntegerParameter>
       <abc:BigIntegerParameter
Name="urn:idmx:3.0.0:systemParameters:dhSubgroupOrder">
           <abc:Value>62...7751</abc:Value>
       </abc:BigIntegerParameter>
       <abc:BigIntegerParameter Name="urn:idmx:3.0.0:systemParameters:dhGen-1">
           <abc:Value>86...3652</abc:Value>
       </abc:BigIntegerParameter>
       <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:systemParameters:dhGen-2">
           <abc:Value>79...3912</abc:Value>
       </abc:Parameter>
   </abc:CryptoParams>
</abc:SystemParameters>
```

2.4.2.2.2 Issuer Parameters

```
<abc:IssuerParameters>
   <abc:MaximalNumberOfAttributes>4</abc:MaximalNumberOfAttributes>
   <abc:HashAlgorithm>sha-256</abc:HashAlgorithm>
   <abc:CryptoParams>
       <abc:PublicKey Version="3.0.0" Technology="urn:idmx:3.0.0:block:sig:cl"
SystemParametersId="0001" PublicKeyId="http://recred.eu/example-issuance">
           <abc:Parameter xsi:type="abc:IntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:maxAtts">
               <abc:Value>4</abc:Value>
           </abc:Parameter>
           <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:base:0">
```

```
                <abc:Value>12...7698</abc:Value>
            </abc:Parameter>
            <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:base:1">
                <abc:Value>34...0982</abc:Value>
            </abc:Parameter>
            <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:base:2">
                <abc:Value>56...8753</abc:Value>
            </abc:Parameter>
            <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:base:3">
                <abc:Value>78..0919</abc:Value>
            </abc:Parameter>
            <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:rsaModulus">
                <abc:Value>35..4672</abc:Value>
            </abc:Parameter>
            <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:base:Z">
                <abc:Value>75...9812</abc:Value>
            </abc:Parameter>
            <abc:Parameter xsi:type="abc:BigIntegerParameter"
Name="urn:idmx:3.0.0:issuer:publicKey:base:S">
                <abc:Value>17...8292</abc:Value>
            </abc:Parameter>
        </abc:PublicKey>
    </abc:CryptoParams>
</abc:IssuerParameters>
```

2.4.2.2.3 Credential Specification

```
<abc:CredentialSpecification Version="Version 1.0" KeyBinding="true"
Revocable="true" xsi:schemaLocation="http://recred7main/resources/xsd/schema.xsd">
<abc:SpecificationUID>http://recred.eu/examples/simple</abc:SpecificationUID>
    <abc:FriendlyCredentialName lang="en">This is just an example with basic
personal information</abc:FriendlyCredentialName>
    <abc:DefaultImageReference>file:/img/simple-example.png</abc:ImageReference>
    <abc:AttributeDescriptions MaxLength="256">
        <abc:AttributeDescription Type="FirstName" DataType="xs:string"
Encoding="urn:abc4trust:1.0:encoding:string:sha-256">
            <abc:FriendlyAttributeName lang="en">Name</abc:FriendlyAttributeName>
        </abc:AttributeDescription>
        <abc:AttributeDescription Type="LastName" DataType="xs:string"
Encoding="urn:abc4trust:1.0:encoding:string:sha-256">
            <abc:FriendlyAttributeName
lang="en">Lastname</abc:FriendlyAttributeName>
        </abc:AttributeDescription>
        <abc:AttributeDescription Type="Birthday" DataType="xs:date"
Encoding="urn:abc4trust:1.0:encoding:date:unix:signed">
            <abc:FriendlyAttributeName
lang="en">Birthday</abc:FriendlyAttributeName>
        </abc:AttributeDescription>
    </abc:AttributeDescriptions>
</abc:CredentialSpecification>
```

2.4.2.2.4 Issued Credential

```
<abc:CredentialDescription RevokedByIssuer="false">
        <abc:CredentialUID>device-cred-1234…78</abc:CredentialUID>
        <abc:FriendlyCredentialName lang="xs:language"> This is just an example
with basic personal information</abc:FriendlyCredentialName>
        <abc:ImageReference>file:/img/simple-example.png</abc:ImageReference>
<abc:CredentialSpecificationUID>http://recred.eu/examples/simple</abc:CredentialSpe
cificationUID>
<abc:IssuerParametersUID>http://recred.eu/exampleissuer/issuance:idemix</abc:Issuer
ParametersUID>
        <abc:SecretReference>secret://software-smartcard-
111111</abc:SecretReference>
        <abc:Attribute>
```

```
            <abc:AttributeUID>-12…9</abc:AttributeUID>
            <abc:AttributeDescription Type="FirstName" DataType="xs:string"
Encoding="urn:abc4trust:1.0:encoding:string:sha-256">
                <abc:FriendlyAttributeName
lang="en">Name</abc:FriendlyAttributeName>
            </abc:AttributeDescription>
            <abc:AttributeValue xsi:type="xs:string">John</abc:AttributeValue>
        </abc:Attribute>
        <abc:Attribute>
            <abc:AttributeUID>-34…9</abc:AttributeUID>
            <abc:AttributeDescription Type="LastName" DataType="xs:string"
Encoding="urn:abc4trust:1.0:encoding:string:sha-256">
                <abc:FriendlyAttributeName
lang="en">Lastname</abc:FriendlyAttributeName>
            </abc:AttributeDescription>
            <abc:AttributeValue xsi:type="xs:string">Doe</abc:AttributeValue>
        </abc:Attribute>
        <abc:Attribute>
            <abc:AttributeUID>xs:ID</abc:AttributeUID>
            <abc:AttributeDescription Type="Birthday" DataType="xs:date"
Encoding="urn:abc4trust:1.0:encoding:date:unix:signed">
                <abc:FriendlyAttributeName
lang="en">Birthday</abc:FriendlyAttributeName>
            </abc:AttributeDescription>
            <abc:AttributeValue xsi:type="xs:string">1940-02-
18Z</abc:AttributeValue>
        </abc:Attribute>
</abc:CredentialDescription>
```

2.4.2.3 IRMA

ReCRED Introduces a Device-Centric Architecture, thus the User ABAC functionalities must be securely and efficiently implemented in mobile devices. The IRMA project [22] has provided an open source Idemix implementation targeted at mobile devices.

The reuse and integration in ReCRED of the card emulator (*irma_android_cardemu*) and the Idemix credential handling operations (*credentials_idemix)* is another effort that is currently ongoing. The *irma_android_cardemu* is an implementation of a credential manager running on Android devices. It manages the storage of issued credentials and the attribute disclosure from them. The *credentials_idemix* component is the IRMA credentials API implementation for Idemix. It provides Java code to represent, handle and perform mathematical operations on credentials.

One of the aims is to use the IRMA card emulator as a cryptographic credential storage as an alternative to the Trusted Execution Environment. The other is the integration of the IRMA credential manager as the ABAC cryptographic credential store on the User Device.

2.4.2.4 Pydemix – Python Idemix Implementation

The above implementation efforts leverage mainly on the Java programming language. This is an advantage for portability, as JVMs are highly available, but it has some drawbacks: a developer needs a non-negligible amount of time to setup her development environment, and studying the software architecture, the relationship between modules and might take even further time.

To tackle these shortcomings we are developing a Python-based Idemix implementation by completing with the sections that are missing (with respect to all the operations that ReCRED needs to support) the partial code base provided by the work in [23].

This prototype implementation has several advantages:

- It allows for fast prototyping of Idemix-based applications
- It is highly portable, as it can be ran both at the server and client/device side
- The cryptographic functions follow strictly the Idemix specification, allowing easy readability and investigation of the correctness of the source code

The prototype source code is currently provided on github [24].

2.5 U-Prove

U-Prove [7] is a cryptographic protocol which assures the user's privacy by minimally disclosing the certified attributes while interacting with an on-line entity. U-Prove has the three actors: the prover (user), the issuer and the verifier. The user to which is issued a cryptographic token interacts with the issuer and the verifier through an issuance and a presentation protocol. The U-Prove token is a container of attributes and it is digitally signed by the issuer entity. The token has a public key and a corresponding private key (both generated at issuance time) which must be kept secret by the prover. The private key can be used non-interactively by signing data which is later verified by the verifier or interactively in the presentation protocol where the prover signs a message in order to prevent replay attacks and demonstrate a proof-of-possession. The usage of the U-Prove token does not reveal the private key, thus mitigating attacks like eavesdropping or replay. The U-Prove technology provides both unlinkability and untraceability because the issuer executes a blind signature over the token and the prover demonstrates the possession of undisclosed attributes by executing a zero-knowledge protocol. Regarding the structure of the U-Prove token, it can contain a TI (Token Information) section and a PI (Prover Information) section. The TI section can be used as a metadata section making the token more informative: it can specify a validity period and is encoded by the Issuer, this section being always disclosed at presentation time. Being always disclosed in the presentation protocol, the PI section is encoded by the prover and is invisible for the issuer at issuance time. Each U-Prove token has a unique identifier, this information being used in identifying repeated visitors of an on-line service or for tracking revoked tokens. The U-Prove token identifier cannot be computed by the issuer, thus preserving the unlinkability and untraceability security attributes. The U-Prove technology permits the usage of a trusted device (smart card, mobile phone or even a trusted third-party server) on the prover side, when issuing a token. The device acts as a U-Prove token extension, having a private key and participating in the presentation protocol. The device can be used to extend multiple U-Prove tokens, even if issued by different issuers. Concerning the repeating visitor scenario the prover can encode a pseudonym which permits the on-line service to recognize him even though using different U-Prove tokens.

2.5.1 U-Prove Primitives

2.5.1.1 Issuer Primitives

The issuer parameters have the following form:

$$\mathrm{UID}_p, desc(G_q), \mathrm{UID}_H, (g_0, g_1, \ldots, g_n, g_t), (e_1, \ldots, e_n), \mathrm{S}$$

UID_p is an octet string that holds an application-specific unique identifier for the Issuer parameters.

$desc(G_q)$ specifies a group G_q of prime order q.

$\text{UID}_H$is an identifier for the hash algorithm.

$(g_0, g_1, \ldots, g_n, g_t)$ is the Issuer public key. To generate g_0, the issuer uses a private key y_0: $g_0 = g^{y_0}$

$(e_1, \ldots, e_n)$ is a list of byte values which state if the corresponding attribute values are hashed when computing the token public key.

S is an application specific octet string for the issuer parameters.

The application specific value n indicates the number of attributes encoded in each token.

Both the Verifier and the Prover must evaluate the Issuer parameters in the following way:

Input

Group description: $desc(G_q)$= (p,q,g)

Public generators: $(g_0, g_1, \ldots, g_n, g_t) \in G_q$

Verify

G_q(if it is a subgroup construction)

p and q are odd prime numbers

q divides p-1

$g \in G_q$and $g \neq 1$

Verify public key elements

for $i \in 0,1,\ldots,n,t$verify that $g_i \in G_q$and $g_i \neq 1$

2.5.1.2 Device Parameters

Credential tokens can be device-protected an when so, the following is the required additional data:

$$g_d, x_d, h_d$$

$g_d \in G_q$is the device generator and must be a generator ofG_q

$x_d \in \mathbb{Z}_q^*$is the device private key

$h_d = g_d^{x_d} \in G_q$is the device public key. This public key is known by the Prover and by the Issuer during the issuance protocol. Both the Prover and the Issuer must verify that the device public key is a valid element of G_q.

2.5.1.3 Token

The token has the following form:

UID_p, h, TI, PI, σ'_z, σ'_c, σ'_r, d

$\text{UID}_p$is an identifier for the issue parameters

$h \in G_q$ is the token public key. It must be an element of G_q

TI denotes the token information field. This section is always disclosed to the Verifier and contains information like token usage restrictions or metadata.

PI is the value of the prover information field. This section contains information asserted by the issuer and is hidden from the Issuer. PI is always revealed during token presentation.

$\sigma'_z \in G_q$And $(\sigma'_c,\sigma'_r) \in Z_q$form the issuer signature

Boolean **d** is used to indicate if token is protected by a device.

2.5.1.4 Token Private Key

The private key of the token is $\alpha^{-1} \in Z_q^*$,α being a secret generated by the Prover in the issuance protocol.

2.5.1.5 Token Public Key

The token public key has the following form:

$$h = \left(g_0 g_1^{x_1} \ldots g_n^{x_n} g_t^{x_t} \left[g_d^{x_d}\right]\right)^{\alpha}$$

$(g_0, g_1, \ldots, g_n, g_t) \in G_q$ is the Issuer's public key.

$\alpha$is a secret value generated by the prover.

$x_t \in Z_q$is computed by hashing the issuance protocol version 0x01, a digest of the issuer parameters and the TI field.

g_d,$x_d$are present if the token is protected by a device.

$x_i \in Z_q$is obtained from the corresponding attribute value $A_i$either by hashing it (if $e_i$is 0x01) or by encoding it directly (if $e_i$is 0x00).

The value of $x_t \in Z_q$is computed in the following way:

Input

Issuer parameter fields: UID_p, $desc(G_q)$, UID_H, $(g_0, g_1, \ldots, g_n, g_t)$, $(e_1, \ldots, e_n)$, S

Token information field: TI

Device protected boolean: **d**

[Device generator: g_d]

Computation

P = H(UID_p, $desc(G_q)$, UID_H, $\langle g_0, g_1, \ldots, g_n, g_t, [g_d]\rangle$, $\langle e_1, \ldots, e_n\rangle$, S)

x_t= H(0x01, P, TI) → Z_q

The values $x_i$are computed in the following manner:

Input

Issuer parameter fields: q, UID_H, e_i

Attribute value: A_i

Computation

If $e_i = 0x01$

If A_i= ∅ then x_i= 0

Else x_i= $H(A_i)$→ Z_q

Else if $e_i = 0x00$

Verify that $0 \leqslant A_i < q$

$x_i = A_i$

Else return error

Return x_i

2.5.1.6 Issuer Signature

The issuer signature is never seen by the Issuer and thus it cannot be used to link a specific token with an issuance protocol. The signature has the following parameters σ_z', σ_c', $\sigma_r'$and is verified in the following way:

Input

Issuer parameter fields: $desc(G_q)$, UID_H, g_0

Token fields: h, PI, σ_z', σ_c', σ_r'

Verification

Verify that $h \neq 1$

σ_c'

Verify that σ_c'= H (h, PI, σ_z', $g^{\sigma_r'} g_0^{-\sigma_c'}$, $h^{\sigma_r'} (\sigma_z')^{-\sigma_c'}$)

2.5.1.7 Token Identifier

The token identifier is computed by hashing the token's public key and the Issuer's signature in the following manner:

Input

Issuer parameter field: UID_H

Token fields: h, σ'_z, σ'_c, σ'_r

Computation

UID_T=H(h,σ'_z,σ'_c,σ'_r)

2.5.2 U-Prove Protocols

2.5.2.1 Issuance Protocol

The issuance protocol requires a precomputation on both sides followed by the exchange of three messages.

The following parameters are common for both the Prover and the Issuer:

- Issuer parameters: $UID_p, desc(G_q, UID_H, (g_0, g_1, \ldots, g_n, g_t), (e_1, \ldots, e_n), S)$
- Attributes: $(A_1, \ldots, A_n), TI$
- Device protected boolean: d
- [Device parameters: g_d, h_d]

2.5.2.1.1 Prover Precomputation

Input:

$x_t := ComputeXt(IP, TI, d, [g_d])$

$x_i := ComputeXi(IP, A_i)$

$\gamma = g_0 g_1^{x_1} \ldots g_n^{x_n} g_t^{x_t} [h_d]$

Prover information field PI

Precomputation:

Generate αat random from Z_q^*

Generate β_1, β_2at random from Z_q

h = γ^{α}

$t_1 = g_0^{\beta_1} g^{\beta_2}$

t_2=h^{β_2}

Compute α^{-1}modq

2.5.2.1.2 Issuer Precomputation

Input:

x_t=ComputeXt$(IP, TI, d, [g_d])$

x_i=ComputeXi(IP, A_i)

$\gamma = g_0 g_1^{x_1} \ldots g_n^{x_n} g_t^{x_t} [h_d]$

Private key: $y_0 \in Z_q$

$\sigma_z = \gamma^{y_0}$

Precomputation:

Generate w at random from Z_q

$\sigma_a = g^w$

$\sigma_b = \gamma^w$

2.5.2.1.3 Exchanged Messages

A) The first message is sent from the Issuer to the Prover: $(\sigma_z, \sigma_a, \sigma_b)$

B) The second message is sent from the Prover to the Issuer: σ_c

$\sigma'_z = \sigma_z^{\alpha}$

$\sigma'_a = t_1 \sigma_a$

$\sigma'_b = (\sigma'_z)^{\beta_1} t_2 \sigma_b^{\alpha}$

$\sigma'_c = H(h, PI, \sigma'_z, \sigma'_a, \sigma'_b) \rightarrow Z_q$

$\sigma_c = \sigma'_c + \beta_1 \text{ mod } q$

C) The third message is sent from the Issuer to the prover: σ_r

$\sigma_r = \sigma_c y_0 + w \text{ mod } q$

D) The Prover generates the token

$\sigma'_r = \sigma_r + \beta_2 \text{ mod } q$

Verify that $\sigma'_a \sigma'_b = (gh)^{\sigma'_r} (g_0 \sigma'_z)^{-\sigma'_c}$

Token T = UID_p, h, TI, PI, σ'_z, σ'_c, σ'_r, d

Private key α^{-1}

In Figure 8 the issuance protocol steps are depicted.

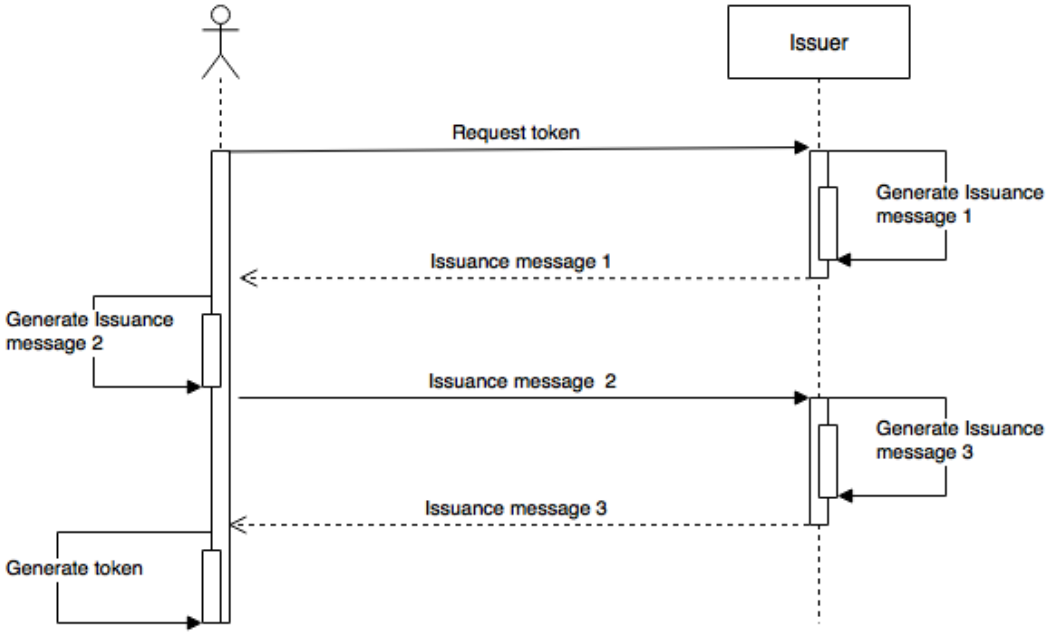

Figure 8: U-Prove Issuance protocol

2.5.2.2 *Presentation Protocol*

In the presentation protocol the Prover sends the token T, the subset of the attributes values it wants to disclose to the Verifier and a presentation proof generated by applying the token private key to a message and the non-disclosed attributes. The presentation proof is used to prove the integrity of the disclosed attributes and to prevent replay attacks.

The presentation protocol can be split in two steps: proof generation and proof verification.

2.5.2.2.1 Proof Generation

This step is a sub-protocol between the device and the Prover.

Device input:

Issuer parameters: $desc(G_q)$, UID_H

Device generator: g_d

Private key: x_d

Prover input:

Issuer parameters: UID_p, desc($desc(G_q)$), UID_H, $(g_0, g_1, \ldots, g_n, g_t)$, $(e_1, \ldots, e_n)$, S

Ordered indices of disclosed attributes: D ⊂ {1, … , n}

Ordered indices of undisclosed attributes: U = {1, … , n} − D

Ordered indices of committed attributes: C ⊂ U

Pseudonym attribute index: p ∈ U ∪ {d}

Pseudonym scope: s

Messages: m, m_d

Token T = UID_p, h, TI, PI, σ'_z, σ'_c, σ'_r, d

Private key: α^{-1}

Attribute values: A_1,…,A_n

[Device generator: g_d]

Compute x_i=ComputeXi(IP,A_i)

Generate random w_0, $[w_d] \in Z_q$

For each i ∈ U generate random $w_i \in Z_q$

A) The first message is sent from the Prover to the device: s

B) The second message is sent from the device to the Prover: a_d, $\left[a'_p, P_s\right]_{p=d}$

Generate w'_dat random from Z_q

a_d=$g_d^{w'_d}$

If $s \neq \emptyset$

g_s=GenerateScopeElement(desc(G_q),s)

a'_p=$g_s^{w'_d}$

P_s=$g_s^{x_d}$

C) The third message is sent from the Prover to the device: c_p,m_d

$a := H\left(h^{w_0}\left(\prod_{i\in U} g_i^{w_i}\right)\left[g_d^{w_d} a_d\right]_d\right)$

If $p \neq \emptyset$and $s \neq \emptyset$

g_s=GenerateScopeElement(desc(G_q),s)

$a_p = H\left(g_s^{w_p}\left[a'_p\right]_{p=d}\right)$

$\left[P_s=g_s^{x^p}\right]_{p\neq d}$

Else $a_p = \emptyset$and $P_s = \emptyset$

For each i ∈ C

Generate σ_i, w_i at random from Z_q

$\tilde{c}_i = g^{x_i} g_1^{\sigma_i}$

$\tilde{\alpha}_i = H\left(g^{w_i} g_1^{\tilde{w}_i}\right)$

UID_T=ComputeTokenID(IP,T)

If p = d then $p' = 0$elsep′=p

$c_p = H\left(UID_T, a, \langle D \rangle, \langle \{x_i\}_{i \in D} \rangle, \langle C \rangle, \langle \{\tilde{c}_i\}_{i \in C} \rangle, \langle \{\tilde{\alpha}_i\}_{i \in C} \rangle, p', a_p, P_s, m\right)$

$c = H\left(\langle c_p, m_d \rangle\right) \rightarrow Z_q$

r_0=cα^{-1}+w$_0$modq

For each i ∈ U, $r_i = -$cx$_i$+w$_i$modq

D) The fourth message is sent from the device to the Prover: r'_d

$c = H\left(\langle c_p, m_d \rangle\right) \rightarrow Z_q$

$r'_d = -cx_d + w'_d mod q$

E) The Prover created the presentation proof:

$[r_d = r'_d + w_d mod q]_d$

For each i ∈ C, $\tilde{r}_i = -c\sigma_i + \tilde{w}_i mod q$

Presentation proof: $\{A_i\}_{i \in D}, a, (a_p, P_s), r_0, \{r_i\}_{i \in U}, [r_d]_d, \{(\tilde{c}_i, \tilde{\alpha}_i, \tilde{r}_i)\}_{i \in C}$

Secret commitment values: σ_i

2.5.2.2.2 Proof verification

Input:

Issuer parameters: UID$_p$, $desc(G_q)$, UID$_H$, $(g_0, g_1, \ldots, g_n, g_t)$, $(e_1, \ldots, e_n)$, S

Ordered indices of disclosed attributes: D ⊂ {1, … , n}

Ordered indices of undisclosed attributes: U = {1, … , n} − D

Ordered indices of committed attributes: C ⊂ U

U-Prove token

Pseudonym attribute index: p ∈ U ∪ {d}

Pseudonym scope: s

Messages: m, m_d

Presentation proof: $\{A_i\}_{i\in D}, a, (a_p, P_s), r_0, \{r_i\}_{i\in U}, [r_d], \{c_i, \alpha_i, \pi_i\}_{i\in C}$

[Device generator: g_d]

Token verification:

VerifyTokenSignature(IP, T)

Presentation proof verification:

$x_t := ComputeXt(IP, TI, d, [g_d])$

For each i ∈ D, x_i=ComputeXi(IP,A_i)

UID_T=ComputeTokenID(IP,T)

If p = d then $p' := 0 else p' := p$

$c_p := H(UID_T, a, \langle D\rangle, \langle\{x_i\}_{i\in D}\rangle, \langle C\rangle, \langle\{c_i\}_{i\in C}\rangle, \langle\{\alpha_i\}_{i\in C}\rangle, p', a_p, P_s, m)$

$c := H(\langle c_p, m_d\rangle) \to Z_q$

Verify that a=H$\left(\left(g_0 g_t^{x_t} \prod_{i\in D} g_i^{x_i}\right)^{-c} h^{r_0} \left(\prod_{i\in U} g_i^{r_i}\right)[g_d^{r_d}]\right)$

If $a_p \neq \emptyset$and $P_s \neq \emptyset$

g_s=GenerateScopeElement(desc(G_q),UID_H,s)

Verify that a_p=H(P_s^c,$g_s^{r_p}$)

For each i ∈ C, verify that $\alpha_i = H(c_i^e g^{r_i} g_1^{\pi_i})$

In Figure 9 are depicted the presentation protocols main steps.

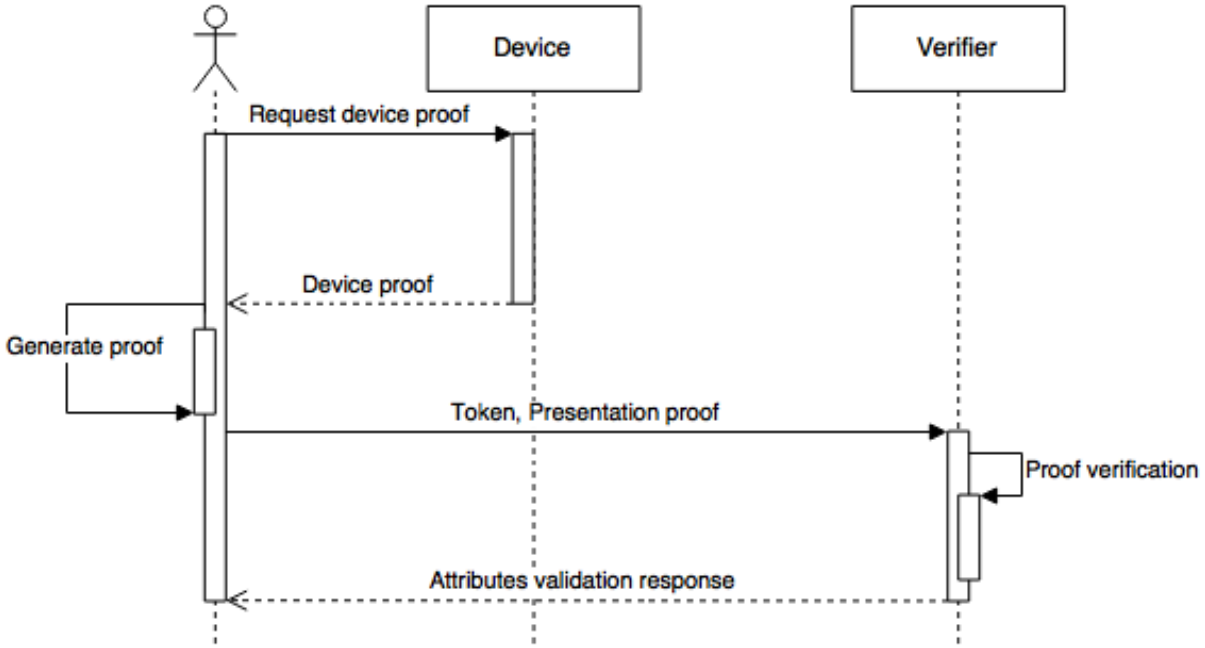

Figure 9: U-Prove Presentation protocol

2.5.3 Implementation

The U-Prove technology implementation can use both subgroups in which is unfeasible to compute discrete logarithms or elliptic curves. Taking this into consideration, U-Prove can be implemented on energy and computationally constrained devices like smart cards or embedded systems. One of the main advantages of the U-Prove cryptographic systems is the fact that it does not infer unnecessary correlation data, in contrast cu classic PKI based solution.

The unlinkability and untraceability properties of the U-Prove protocol can be bypassed only by application-specific attributes disclosed in the presentation protocol. Even though the U-Prove protocol presents the verifier, issuer, prover and device as separated entities, multiple roles can be played by one entity of a role can be distributed to multiple entities without breaking the U-Prove security.

Another advantage of the U-Prove technology is the fact that it permits the usage of extensions, thus being suitable for multiple application scenarios mainly because it supports custom modifications. With the aid of U-Prove extensions, features like revocation, advanced predicate proofs on the attributes or even identity escrow can be added.

2.5.4 Interfaces

In order for the U-Prove protocol to be used in miscellaneous application scenarios, like web service enabled applications, its cryptographic primitives must be serialized. Regarding the parameter serialization we propose the XML format for compatibility with already existing attribute based web service formats, like the one proposed by the ABC4Trust [41] project. For the serialization of the U-Prove primitives we had as a reference Microsoft U-Prove proposal the WS-Trust specifications [40].

In the following serialization XML schema the ds:CryptoBinary element is an extension of the xs:base64Binary element. The ds:CryptoBinary element is used for the serialization of mathematical elements: the binary value is conditionally zero-extended to make its length multiple of 8 bits and then it is base64 encoded.

2.5.4.1 Issuer Parameters

The issuer parameters have the following form:

UID_p, (p,q,g), UID_H, $(g_0, g_1, \ldots, g_n, g_t)$, $(e_1, \ldots, e_n)$,$(z_0, z_1, \ldots, z_n, z_t)$, S

The issuer parameters contain an additional $z_d$if device protected tokens are supported.

The serialized issuer parameters have the following form:

```
<xs:IssuerParameters Version="1.0" N="xs:unsignedShort">
 <xs:UIDP>xs:anyURI</xs:UIDP>
  <xs:Group>
   <xs:GroupDescription GroupType=””>
    <xs:P>ds:CryptoBinary</xs:P>
    <xs:Q>ds:CryptoBinary</xs:Q>
    <xs:Gen>ds:CryptoBinary</xs:Gen>
   </xs:GroupDescription>
  </xs:Group>
  <xs:Gvalues GeneratorsName="xs:anyURI">
   <xs:G>ds:CryptoBinary<xs:G>+
 </xs:Gvalues>
 <xs:Evalues>
  <xs:E>xs:unsignedbyte</xs:E>*
 </xs:Evalues>
 <xs:Zvalues>
  <xs:Z>ds:CryptoBinary</xs:Z>+
 </xs:Zvalues>
 <xs:Zd>ds:CryptoBinary</xs:Zd>?
 <xs:Specification>xs:anyURI</xs:Specification>
</xs:IssuerParameters>
```

`/xs:IssuerParameters/xs:UIDP`

This element contains an URI which identifies the parameters. It is UTF-8 encoded.

`/xs:IssuerParameters/xs:Group`

This element contains the mathematical components of the group.

`/xs:IssuerParameters/xs:Group/xs:GroupDescription`

This element contains the mathematical elements (p,q,g).

`/xs:IssuerParameters/xs:Group/xs:GroupDescription/xs:P`

This element contains the base64 encoding of element P.

`/xs:IssuerParameters/xs:Group/xs:GroupDescription/xs:Q`

This element contains the base64 encoding of element Q.

`/xs:IssuerParameters/xs:Group/xs:GroupDescription/xs:Gen`

This element contains the base64 encoding of element G.

`/xs:IssuerParameters/xs:GValues`

This element contains the G values $(g_0, g_1, \ldots, g_n, g_t)$.

`/xs:IssuerParameters/xs:Gvalues/xs:G`
This element contains the base64 encoding of the G values.

`/xs:IssuerParameters/xs:Evalues`
This element contains the values $(e_1, \ldots, e_n)$.

`/xs:IssuerParameters/xs:Evalues/xs:E`
This element contains an E value.

`/xs:IssuerParameters/xs:Zvalues`
This element contains the values $(z_0, z_1, \ldots, z_n, z_t)$.

`/xs:IssuerParameters/xs:Zvalues/xs:Z`
This element contains the base64 encoding of each Z value.

`/xs:IssuerParameters/xs:Zd`
This element contains the base64 encoding of the $z_d$value.

2.5.4.2 Token

The U-Prove token has the following form:

$$UID_p, h, TI, \sigma_z', \sigma_c', \sigma_r', d$$

The serialized U-Prove token has the following form:

```
<xs:Token Version="1.0">
 <xs:UIDP>xs:anyURI</xs:UIDP>
 <xs:H>ds:CryptoBinary</xs:H>
 <xs:TI>
  <Gd GdName=”xs:anyURI”/>?
 </xs:TI>
 <xs:SigmaZPrime>ds:CryptoBinary</xs:SigmaZPrime>
 <xs:SigmaCPrime>ds:CryptoBinary</xs:SigmaCPrime>
 <xs:SigmaRPrime>ds:CryptoBinary</xs:SigmaRPrime>
</xs:Token>
```

`/xs:Token/xs:UIDP`
This element contains an URI which identifies the issuer parameters

`/xs:Token/xs:H`
This element contains the bas64 encoding of value h.

`/xs:Token/xs:TI`
This element contains the Token Information field.

`/xs:Token/xs:TI/xs:Gd`
This element contains the device generator details if the token is device protected. The attribute contains an URI which specifies the device generator details.

`/xs:Token/ xs:SigmaZPrime`
This element contains the base64 encoding of value σ_z'

`/xs:Token/ xs:SigmaCPrime`
This element contains the base64 encoding of value σ_c'

`/xs:Token/ xs:SigmaRPrime`
This element contains the base64 encoding of value σ_r'

2.5.4.3 Protocols Serialization

2.5.4.3.1 Issuance Protocol Serialization

Regarding the issuance protocol there are 3 U-Prove messages along with a client initial request, which are serialized in the following manner:

0. The initial request is sent from the prover to the issuer. The initial request contains the device parameters if the token is device-protected.

```
<xs:DeviceParameters Id=”xs:anyURI”>
 <xs:Gd GdName=”xs:anyURI”/>
 <xs:Hd>xs:cryptoBinary</up:Hd>
 <xs:Signature>xs:base64Binary</xs:Signature>
</xs:DeviceParameters>?
```

```
/xs:DeviceParameters
```
This element contains the device parameters.

```
/xs:DeviceParameters/xs:Gd/@GdName
```
This attribute contains the device generator parameter defined by an URI.

```
/xs:DeviceParameters/xs:Hd
```
This element contains the base64 encoding of the device public key h_d.

```
/xs:DeviceParameters/xs:Signature
```
This element contains the base64 encoding of an RSA signature which authenticates the device parameters. The Issuer must check this value in order to proceed with the issuance protocol.

2.5.4.3.1.1 Message sent from issuer to prover:

```
<xs:IssuanceMessage1>
 <xs:SigmaPair>
  <xs:SigmaA>ds:cryptoBinary</xs:SigmaA>
  <xs:SigmaB>ds:cryptoBinary</xs:SigmaB>
 </xs:SigmaPair>+
</xs:IssuanceMessage1>
```

```
/xs:IssuanceMessage1
```
This element contains the first issuance protocol message.

```
/xs:IssuanceMessage1/xs:SigmaPair
```
This element contains a the mathematical components of the first issuance protocol message.

```
/xs:IssuanceMessage1/xs:SigmaPair/xs:SigmaA
```
This element contains the base64 encoding of the value σ_a.

```
/xs:IssuanceMessage1/xs:SigmaPair/xs:SigmaB
```
This element contains the base64 encoding of the value σ_b.

2.5.4.3.1.2 Message sent from the prover to the issuer

```
<xs:IssuanceMessage2>
 <xs:SigmaC>ds:cryptoBinary</xs:SigmaC>+
</xs:IssuanceMessage2>
```

`/xs:IssuanceMessage2`
This element contains the second issuance protocol message.

`/xs:IssuanceMessage2/xs:SigmaC`
This element contains the base64 encoding of the value σ_c.

3. The third message is sent from the issuer to the prover:

```
<xs:IssuanceMessage3>
 <xs:SigmaR>ds:cryptoBinary</xs:SigmaR>+
</xs:IssuanceMessage3>
```

`/xs:IssuanceMessage3`
This element contains the third issuance protocol message.

`/xs:IssuanceMessage3/xs:SigmaR`
This element contains the base64 encoding of the value σ_r.

2.5.4.3.2 Presentation protocol serialization

The presentation protocol has 2 phases: the proof generation and the proof verification. If the token is device-protected then the prover must interact with the device for the proof generation. The proof generation phase implies the following serialization:

```
<xs:Token>...</xs:Token>
<xs:SubsetProof>
 <xs:A>xs:base64Binary</xs:A>
 <xs:R0>ds:cryptoBinary</xs:R0>
 <xs:Responses>
  <xs:R>ds:crypto64Binary</xs:R>+
 </xs:Responses>?
 <xs:Rd>ds:cryptoBinary</xs:Rd>?
</xs:SubsetProof>
```

`/xs:Token`
This element contains the serialization of the U-Prove token.

`/xs:SubsetProof`
This element contains the token proof. The mathematical components are:

$$A_i(i \in D), a, r_0, r_i(i \in U)$$

`/xs:SubsetProof/xs:A`
This element contains the base64 encoding of value a.

`/xs:SubsetProof/xs:R0`
This element contains the base64 encoding of value r_0.

`/xs:SubsetProof/xs:Responses`
This element contains the responses for the undisclosed attributes (index order must be preserved).

`/xs:SubsetProof/xs:Responses/xs:R`
This element contains the base64 encoding of a response.

`/xs:SubsetProof/xs:Rd`

This element contains the base64 encoding of the value r_d. This value is present only if the token is device protected.

2.5.4.3.3 Serialization Considerations

The serialization framework defined by the ABC4Trust project is cryptographic protocol agnostic, thus being compatible with the U-Prove implementation details. Regarding the issuer parameters define by the ABC4Trust project the <abc:CryptoParams> elements will be implemented with the aid of the element <xs:IssuerParameters>.

The issuance protocol defined by U-Prove requires 3 specific messages which follow an initial user request. The ABC4Trust issuance protocol framework specifies a variable number of steps where the final step has as result an issued credential.

The ABC4Trust project proposes the usage of an abc:IssuanceMessage for wrapping each issuance message protocol. The abc:IssuanceMessage must contain either an issuance policy, issuance token or application-specific cryptographic data. Thus the elements xs:IssuanceMessage1, xs:IssuanceMessage2 and xs:IssuanceMessage3 will be wrapped in an abc:IssuanceMessage element.

The U-Prove presentation protocol requires the deliver of a presentation token which contains among others, a proof of possession for the undisclosed attributes. The U-Prove cryptographic material defined by the elements xs:Token and xs:SubsetProof can be wrapped in the abc:CryptoEvidence defined in the abc:PresentationToken.

2.5.5 Prototype

2.5.5.1 U-Prove WS

The U-Prove server entities (Verifier and Issuer) communicate with the rest of the ReCRED components by means of web services. The ReCRED components query the U-Prove WS entity in order to obtain U-Prove specific cryptographic serialized components. After querying the U-Prove WS entity the ReCRED components will wrap the the cryptographic elements in a protocol agnostic format defined by the ABC4Trust project.

U-Prove WS will implement the following methods:

IssuerParameters getIssuerParameters()

This method will return the issuer parameters.

Path	/issuer/getIssuerParameters
HTTP Method	GET
Input type	application/xml
Output Type	text/xml
Output Format	IssuerParameters

IssuanceMessage1 startIssuanceProtocol(DeviceParameters deviceParams)

This method is called when user wants to start an issuance protocol. If the U-Prove issuer returns an IssuanceMessage1 element then the issuance protocol will continue.

Path	/issuer/startIssuanceProtocol
HTTP Method	POST
Input type	application/xml
Input format	DeviceParameters
Output Type	Text/xml
Output Format	IssuanceMessage1

IssuanceMessage3 finishIssuanceProtocol(IssuanceMessage2 message2)

This method is called in order to finish the issuance protocol. If the U-Prove issuer returns an IssuanceMessage3 element then the issuance protocol finished succesfully.

boolean validateToken(Token token, SubsetProof proof)

This method is called in order to validate a U-Prove token. If the U-Prove token returns a true value the disclosed attributes were successfully validated. Because a web service can have only one root element the following element will comprise both Token and SubsetProof.

```
<TokenAndSubsetProof>
 <Token>...</Token>
 <SubsetProof>...</SubsetProof>
</TokenAndSubsetProof>
```

The boolean value is defined by the following element:

```
<recred:Boolean value=”xs:boolean”>
```

Path	/verifier/validateToken
HTTP Method	POST

Input type	application/xml
Input format	TokenAndSubsetProof
Output Type	text/xml
Output Format	Boolean

2.5.5.2 U-Prove Prototype Implementation

Regarding the U-Prove implementation, Microsoft open-sourced a C# and JavaScript implementation, while having also a legacy Java version. In order to preserve the compatibility with the other ReCRED technologies the U-Prove implementation will be exposed as a web-service. The ReCRED protocol agnostic entity will communicate with the U-Prove WS by means of XML messages which follow the schema proposed in the U-Prove serialized elements, thus stripping the protocol wrapping proposed in the ABC4Trust project. Concerning the U-Prove server side entities: issuer and verifier, the C# implementation will be used. On the prover side (mobile device) it will be implemented a Java based version of U-Prove user and device functionality for Android. For other technologies a C++ version will also be considered. The U-Prove device functionality will be exposed to the other applications by means of an Android service. The U-Prove specifications stress the importance of a secure channel between the user and the device, thus the device Android service will establish a secure communication channel with third-party applications.

ABC4Trust offers an implementation of an agnostic privacy-preserving attributed based credential engine which is protocol agnostic: the p2abc project. The FIWARE project built an additional layer on top of the p2abc project offering a privacy FIWARE GE (Generic Enabler): the p2abc-zhaw project. The latter project use the Microsoft C# based U-Prove implementation. Both ABC4Trust and FIWARE can be used as web services and the implementation is Java based (the deployment environment could be Tomcat or another servlet engine).

2.6 Attribute-Based Encryption (ABE)

Attribute Based Encryption is an emergent new kind of asymmetric cipher. Similar to ordinary public key encryption schemes, a content is encrypted using a public key which does not reveal any information useful to decrypt the data, e.g. the private key. However, unlike ordinary public key schemes, the decryption key is not unique, but multiple users, having different keys, may decrypt a same message. Furthermore, and distinguishing feature of ABE, a user can decrypt a message only if the user is provided with a set of attributes which satisfy a given policy.

We are specifically interested in a variant of ABE called Ciphertext Policy, CP-ABE, where the policy which needs to be satisfied by the user’s attributes is directly integrated in the encrypted data itself, hence it “travels” with the data. Note that the combination of encryption (for confidentiality) and policy (for access control) in CP-ABE appears to be an extremely convenient approach for services that requires to release a specific resource outside of the trusted limits. This is best understood going into an example scenario using CP-ABE as cryptographic technique.

Let us assume that a content provider P wishes to encrypt a message M for a given set of users without the need to know a priori the identity of each individual user which shall be able to access the data, but wants to permit access to the data only to users which satisfy a given policy Π expressed in terms of attributes associated to the end users. Such a policy can be any arbitrary combination of "AND" and "OR" conditions, for instance

$$\Pi = \text{(italy:citizen AND job:executive) OR (job:doctor)}$$

Notably, at encryption time, the content provider only requires to know:

- the subset of attributes of interest, which are ordinary natural language strings
- the public key of the authority which has issued such attributes (as discussed later, CP-ABE was recently extended to operate with multiple non-coordinating authorities).

Not only CP-ABE does *not* require the content provider to a priori know the set of users which will be able to access the message, but it completely decouples the encryption process from the management of the user attributes. Indeed, suppose that CP-ABE encryption of a message M using policy Π occurs at a given time, say t_1. Let $E[\Pi,M]$ be the resulting ciphertext, where we use a notation which highlights the fact that the policy Π is indeed integrated in the encrypted data itself. Suppose now that, at a subsequent time $t_2>t_1$, a new user, say U_x, needs to be added to the set of users allowed to access the message. Such user just need to offline and once for all access the attribute-issuing authority (or the set of involved authorities in a multi-authority scenario) to be provided with the attribute that permits the user to decrypt the message, in the meantime cached in the network or replicated in an untrusted storage, at any later time.

2.6.1 A multi-authority CP-ABE architecture

The concept of CP-ABE has been originally introduced by Bethencourt, Sahai and Waters in 2007 [18]. This first construction however had a significant practical limitation in the fact that attributes were issued by a single, global, authority. In order to overcome such a limitation, the cryptographic community attempted to devise multi-authority CP-ABE schemes, with the first proposal in this field being a paper by Chase [19]. However, this first multi-authority proposal, as well as the subsequent extensions, still required some form of cooperation (at least offline) among the authorities. In the real world, such form of cooperation is deemed to be unviable, as it would force all possible authorities (ranging from banks, governments, visa offices, and even individuals) to interact at least once each other, as well as re-run a cooperation protocol every time a new authority is deployed. Also (mostly?) for this reason, CP-ABE did not have any notable practical success outside the restricted community of cryptographic researchers.

In a breakthrough paper, dated 2011 [20], Lewko and Waters proposed the first fully decentralized CP-ABE construction, thus broadly extending CP-ABE's application range and make it fitting the real world needs of large scale networks and deployments. In this context, fully decentralized means that access policies can be specified over an arbitrary set of attributes issued by multiple independent and not cooperating authorities (possibly not even knowing each other's existence). The far from being trivial technical challenge solved in [20] was the construction of a scheme resistant to collusion among users; in other words, if user U_1 holds attribute $attr_1$ issued by an authority A_1 and user U_2 holds attribute $attr_2$ issued by a *different* authority A_2 which has never cooperated or exchanged any information with A_1, and even if the two users collude by exchanging their secrets associated to such

attributes, as well as any other possible information locally held by the two users, it should be impossible (computationally hard) for each of these users to decrypt a message encrypted with the policy $\Pi = (\text{attr}_1 \text{ AND } \text{attr}_2)$. We refer the reader to the original work [20] for the cryptographic construction details. Despite the original construction [20] still suffers of some minor technical limitations, we believe that the notion of independent and fully decentralized authority therein exploited very well fits with the real world needs.

Motivated by the availability of an actual, fully decentralized, multi-authority CP-ABE cryptographic construction, in what follows we preliminary sketch a multi-authority CP-ABE-based security architecture.

Attribute-issuing authorities. An authority A_i is any *arbitrary* entity (hence even including individual users) which autonomously decides to issue attributes. The set-up of an authority is thus an independent decision, and does not require any coordination or interaction with a global authority. The only requirement an authority must adhere is to use a same set of globally-defined and publically known system parameters (in essence, a small set of standardized parameters, which, to make an illustrative example for the the specific CP-ABE setting of [20], appendix D, simply consist in a bilinear group $\mathbb{G}$ of prime order *p*, in a generator *g* of the prime order group, and in a hash function *H* mapping global identity names into points of the group $\mathbb{G}$). An authority *x* will be characterized by a pair of keys: a public key $A_{x,PK}$, and an associated secret key $A_{x,SK}$. An authority is univocally identified by its public key: since this public key cannot be decided by the authority, but is computed by a cryptographic algorithm, the possibility that two authorities shall have the same PK is negligible. Although not technically necessary, if human readable names shall be used for authorities, an ordinary PKI must be supplementary used to bind the authority's public key to its real world name, and avoid authority impersonation attacks. More formally, we summarize the set up of an authority with a publically known algorithm:

$$A_x.\text{AUTHORITY\_SETUP}(\text{global parameters}) \rightarrow A_{x,PK}, A_{x,SK}$$

which is independently run by each authority, and which computes the authority's public and private key pair.

Attributes. An attribute is a plain text string defined by, and associated to, an authority. For instance, an attribute can be as general as the string "visa" associated to a country-wide immigration authority and used to grant access permissions to a given country, or as specific as the string "office-mate" issued by an individual. Attributes shall not need to be globally unique (thus simplifying naming issues), but just need to be unique inside a same authority. For an example, two countries (say Russia and Japan) can issue a same attribute string named "visa", but these two attributes are different as they are issued by different authorities. Whenever ambiguity occurs, we will use the scope symbol ":" to differentiate the two attributes, e.g. Russia:visa versus Japan:visa, but we stress that this is just a notational convenience and not the bit string associated to the actual attribute (which, in both cases, it is simply the string "visa").

Attribute-issuing procedure and Global identity names. In order to get an attribute from an authority, an user must have a global identity name, called UID (user ID), which must be a globally unique bit-string, for instance, an email address. Attributes issued to different identity names (even if belonging to a same human user, e.g. two different email addresses) will not be combined in a same access

control policy. For instance, if a same human user holds two identity names, e.g. foo@mail.com holding attribute x and foo@recred.com being issued attribute y, the user will *not* be able to access a data encrypted with CP-ABE using the policy (x AND y). In order to be granted an attribute, a user will offline submit to an authority its global identity name, and if the authority decides to issue the required attribute, the user will receive back a secret key uniquely associated to both the user as well as the attribute. Note that this implies that different users will get different secrets for a same attribute. Formally, we summarize the attribute issuing procedure as an algorithm

$$A_x.\text{ATTRIBUTE\_ISSUING}(\text{UID}, \text{attr\_j}, A_{x,SK}) \rightarrow K_{UID,attr\_j}$$

Where UID is the global identity name of the user, attr_j is the issued attribute name, $A_{x,SK}$ is the Authority secret key, and $K_{UID,attr\_j}$ is the secret key released to the user for the considered attribute. This algorithm shall be executed by the authority, and the resulting secret key shall be provided to the user via a secure channel.

Note that the compelling aspect of the above sketched architecture resides in the fact that it does not specify any necessary system component (e.g. unlike IPsec, where security associations require to be supported by security association databases and security policy databases). The trust model underlying the access control operation is mandated to individual trust relations (which can eventually, but not necessarily, exploit a certification PKI infrastructure) among entities and attribute-issuing authorities, rather than to a trust infrastructure. This can be very clearly highlighted through the following encryption use-case example. Assume that an user U_x decides to share a message M encrypted with the policy

$$\Pi = (\text{Italy:citizen AND age:greater\_than\_18 AND } U_x\text{:friend}) \text{ OR (italian\_police:officer)}$$

where attributes are written using scope notation (i.e., authority:attribute). In order to encrypt message M, the user needs to decide/have:

- the attribute bit strings, i.e. “citizen”, “greater_than_18”, “friend”, and “officer”;
- the access control policy;
- the public keys of the four involved authorities, i.e.
    - a national authority from Italy which releases citizenship permissions;
    - an authority which certifies, by issuing a relevant attribute, that an user has an age greater than 18;
    - a national police authority, and
    - the user herself; indeed, since any entity can become authority, the user can as well decide to issue her own attributes, such as the “friend” attribute highlighted in the policy.

Once the message is encrypted, the user knows that the message will be accessed only by other users which have been issued a set of attributes by the specific authorities considered. Indeed, the encryption of a message is performed by ciphering the message using the attribute bit strings as well as the public keys of the relevant authorities which are in charge of issuing the given attributes. Note that this is a significant generalization of the ordinary asymmetric public key encryption, with the notable difference that the public key used during encryption is not anymore the one of the recipient of the message, but are those of the attribute issuing authorities. In essence, in terms of trust, CP-ABE

implies that the user just relies on her individual trust in the *specific* authorities involved, which are identified through their public keys.

Since Attribute Based Encryption schemes realize an implicit access control mechanism on the encrypted data, we believe that the ReCRED ABAC architecture could benefit of the usage of such techniques. Indeed it can be used both to realize an access control on static data distributed in the network (data encryption) both an access control for the user (token encryption).

2.7 Integration of Idemix and U-prove in TEE

The Android Keystore and the relative API [26] provide a secure way to handle keys and key-related cryptographic function with the help of a Trusted Execution Environment (TEE) if one is provided by the device. By the time this document is written we cannot run arbitrary code in the TEE in the form of a trusted application as these applications are pre-installed by hardware and/or device vendors and come bounded with device’s processor. We made some initial contacts with vendors in an attempt to find a way to implement part of the U-Prove and Idemix protocols inside the TEE as a trusted application.

For the time being, the TEE will only be used for encryption of U-Prove and Idemix cryptographic credentials for secure storage and we will continue the investigation of leveraging security of the protocols through a custom trusted application. Please note that we have comprehensively analyzed the TEE technology in the Deliverables 2.3 and 3.1 of WP2 and WP3 respectively.

For Idemix and U-Prove, the credentials will be encrypted and stored by the device as soon as they are issued and only be decrypted and used when certain user’s attributes need to be revealed and verified by a third party. The encryption key will be a 256 bit key and will be used along with Advanced Encryption Standard (AES) algorithm. Moreover the key will bound to the specific encryption algorithm and block mode as well as the purposes of encryption and decryption. The key will be generated with the help of Android’s KeyGenerator class [27]. A key generation wrapper function is presented as an example below.

```
protected void createSecureStorageKey() {
    try {
        KeyGenerator secureStorageKey = KeyGenerator.getInstance(
                KeyProperties.KEY_ALGORITHM_AES, "AndroidKeyStore");

        secureStorageKey.init(new KeyGenParameterSpec.Builder(
                "SecureStorageKey",
                KeyProperties.PURPOSE_ENCRYPT | KeyProperties.PUR-
POSE_ENCRYPT)
                .setBlockModes(KeyProperties.BLOCK_MODE_CBC)
                .setKeySize(256)
                .build());

        secureStorageKey.generateKey();
    } catch (Exception e) {
        Log.e("ENCRYPTION_LOG_TAG", "I got an error", e);
    }
}
```

As we have already mentioned the ciphering will be performed in AES and more specifically using Cipher Block Chaining (CBC) mode and PKCS7 padding mode. The ciphering will be performed with the help of Android's Cipher class [28] which provides access to cryptographic implementations inside the TEE. The instantiation of the ciphering class for the selected encryption scheme will be performed just like the example below.

```
Cipher c = Cipher.getInstance("AES/CBC/PKCS7Padding");
```

2.8 Future Work

What we presented here represents the starting point of the prototyping of components for the Attribute Based Access Control architecture of the ReCRED framework. Such prototypes allow to start soon with the integration of such components so that the ABAC infrastructure can be deployed and exploited by the ReCRED project. Features that should be included in the future ABAC architecture include:

- **ABE**: Attribute Based Encryption schemes allow to realize implicit attribute based access control on resources only by encrypting them by means of a Boolean policy over attributes. We believe that the ReCRED architecture should exploit such feature in order to obtain an infrastructure-less ABAC mechanism alternative to standard ones.
- **TEE**: Since the ReCRED project is strongly focused on Device Centric Authentication, we believe that the ABAC infrastructure should be integrated securely with the user's device. In order to reach such target, we are investigating new TEE platforms that will enable the implementation of ABAC algorithms (i.e. Idemix, U-Prove and ABE) in a trusted and tamper-proof device (§ Section 2.7).
- **Backup procedures**: The Backup Protocol is run between the User Device and the Identity Consolidator or between the Issuer and the Identity Consolidator. It allows the secure transfer of credentials from to the Identity Consolidator, which can store them securely or, if specifically requested, act on behalf of the User.

- **Revocation procedures**: Each one of the ABAC mechanisms presented above provides its own mechanism for the revocation of credentials/attributes. In order to realize a realistic and practical platform for the access control, we will investigate and deploy such revocation mechanism to be integrated in the ReCRED framework and procedures.

3 Attributes and Policies for Attribute-Based Access Control

3.1 The ReCRED Definition of the Attributes

The attributes are identity characteristics of the users. For example, the full name, the age and the mobile phone number of the users. These identity characteristics take some values, which are called attribute values. For example, John Adams is a valid value for the full name attribute, 18 is a valid value for the age attribute and 96873446 is a valid value for the mobile phone number attribute.

3.2 Access Control Policies

Access Control Policies are high-level requirements that specify how access is managed and who can access specific resources and with which permissions (actions). ABAC allows an unrestricted number of user attributes to be combined to satisfy a set of access control policies.

The XACML (eXtensible Access Control Markup Language) standard defines a general-purpose access control policy language implemented in XML. XACML is the widely used authorization mechanism for web services.

3.2.1 The advantages of XACML

1. It is standard. This means that has been reviewed by a large community of experts and users, so we don’t need to think about all the complex issues involved in designing a new policy language.
2. It is generic. This means that a policy can be used in any environment, by many different kinds of applications.
3. It is powerful. This means that the standard language supports a wide variety of data types, functions and rules about combining the different results of different policies.

Each XACML document contains a Policy or PolicySet root XML tag. Because a Policy or a PolicySet may contain multiple rules or policies respectively, XACML uses a collection of Combining Algorithms to reconcile the multiple decisions. There are Policy Combining Algorithms which are used by PolicySets and Rule Combining Algorithms which are used by Policies.

For Example:

If there are two rules (or policies) that contradict each other then the Combining Algorithm is responsible to resolve the conflict.

Rule1: “If a user is a professor, belongs to the Electrical Engineer Department”, he can get access to the Electrical Engineer Lab network with the permission to print.

Rule2: “No one can print before 9:00 AM”.

Request: “John wants to print at 8:00 AM”

So the Combining Algorithm is responsible to decide which of the two rules wins.

Examples of Combining Algorithms:

Permit-Override Algorithm: If at least one rule (policy) is evaluated as “Permit” the integrated decision will also be “Permit”.

Deny-Override Algorithm: If at least one rule (policy) is evaluated as "Deny" the integrated decision will also be "Deny".

3.2.2 The XACML Policy Structure

A **<Policy>** represents a single access control policy and it contains a set of **<Rule>** elements and a rule combining algorithm. A **<Rule>** element contains a **<Target>** element which is a set of **<Subject>** elements, **<Resource>** elements, **<Action>** elements and **<Environment>** elements. It is also contains an **<Effect>** element and a **<Condition>** element.

```
<Policy PolicyId="" RuleCombiningAlgId="">
<Rule RuleId="" Effect="">
   <Target>
      <Subjects>…</Subjects>
      <Resources>…</Resources>
      <Actions>…</Actions>
      <Environments>…</Environments >
   </Target>
   <Condition FunctionId="">…</Condition>
</Rule>
</Policy>
```

<Target> element:

This element shows a set of simplified conditions for the Subject, the Resource and the Action. The conditions must be met for a PolicySet, a Policy or a rule to apply to a request. Once a Policy or a PolicySet has been found to apply to a request, its rules are evaluated in order to determine the decision. PolicySet, Policy and rule can all contain Target elements.

<Subjects> element:

This element shows who is requested access to a resource and may contain one or more **<Subject>** elements. Within the **<Subject>** element a sub-element called **<SubjectMatch>** is used in order to define the logic to match subject elements. A **<SubjectMatch>** element contains the name and the value of the subject attribute. XACML uses **<SubjectAttributeDesignator>** to retrieve subject attribute values from a request and **<SubjectAttributeSelector>** to retrieve subject attribute values from an XPath query.

<Resources> element:

This element shows the requested resource and may contain one or more **<Resource>** elements. Within the **<Resource>** element a sub-element called **<ResourceMatch>** is used in order to define the logic to match resource elements. A **<ResourceMatch>** element contains the name and the value of the resource attribute. XACML uses **<ResourceAttributeDesignator>** to retrieve resource attribute values from a request and **<ResourceAttributeSelector>** to retrieve resource attribute values from an XPath query.

<Actions> element:

This element shows the type of access of a user to a resource and may contain one or more **<Action>** elements. Within the **<Action>** element a sub-element called **<ActionMatch>** is used in order to define the logic to match action elements. An **<ActionMatch>** element contains the name and the value of the action attribute. XACML uses **<ActionAttributeDesignator>** to retrieve action attribute values from a request and **<ActionAttributeSelector>** to retrieve action attribute values from an XPath query.

<Environments> element:

This element is used to define system attributes which effects the authorization and may contain one or more **<Environment>** elements. Within the **<Environment>** element a sub-element called **<EnvironmentMatch>** is used in order to define the logic to match environment elements. An **<EnvironmentMatch>** element contains the name and the value of the environment attribute. XACML uses **<EnvironmentAttributeDesignator>** to retrieve environment attribute values from a request and **<EnvironmentAttributeSelector>** to retrieve environment attribute values from an XPath query.

<Condition> element:

This element specifies the authorization logic and always contains a Boolean expression. For example an expression of attribute values.

<Effect> element:

This element shows the decision of the rule if its **<Condition>** element is True.

For Example:

“All the users can get access to the Electrical Engineer Lab network with the permission to print, between 10:00 AM and 3:00 PM”.

Subjects: Any subject

Resources: Electrical Engineer Lab network

Actions: Print

Environment: Current Time between 10:00 AM and 3:00 PM

Effect: Permit

Condition: Based on the environment attributes (Time between 10:00 AND 15:00)

```
<Policy PolicyId="Policy1" RuleCombiningAlgId="Permit-Overrides">

<Rule RuleId="1" Effect="Permit">
   <Target>
     <Subjects>
         <AnySubject/>
     </Subjects>
      <Resources>
         <Resources>
         <ResourceMatch
            MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue
            DataType="http://www.w3.org/2001/XMLSchema#string">Electrical
                     Engineer Lab network
         </AttributeValue>
         <ResourceAttributeDesignator
           AttributeId="urn:oasis:names:tc:xacml:1.0:resource:resource-id"
           DataType="http://www.w3.org/2001/XMLSchema#string"/>
         </ResourceMatch>
        </Resource>
     </Resources>
     <Actions>
         <Action>
         <ActionMatch
          MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
         <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#string">print
         </AttributeValue>
         <ActionAttributeDesignator
          AttributeId="NetworkAction"
          DataType="http://www.w3.org/2001/XMLSchema#string"/>
         </ActionMatch>
         </Action>
     </Actions>
  </Target>
```

```
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:and">

      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-
       greater-than-or-equal"
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
       time-one-and-only">

      <EnvironmentAttributeSelector
       AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-
       time"
       DataType="http://www.w3.org/2001/XMLSchema#time"/>

       </Apply>
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#time">10:00:00
        </AttributeValue>
       </Apply>


      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:time-
       less-than-or-equal"
      <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:
       time-one-and-only">

      <EnvironmentAttributeSelector
       AttributeId="urn:oasis:names:tc:xacml:1.0:environment:current-
       time"
       DataType="http://www.w3.org/2001/XMLSchema#time"/>

       </Apply>
        <AttributeValue
          DataType="http://www.w3.org/2001/XMLSchema#time">15:00:00
        </AttributeValue>
       </Apply>

</Condition>
```

```
</Rule>

</Policy>
```

4 Application Scenarios

The goal of these application scenarios is to demonstrate the availability and viability of the proposed ABAC architecture in real-world examples. The proposed ABAC architecture will deliver ease and efficiency to applications thus enhancing the overall user experience. Ideally, these application scenarios will result in the creation of true commercial services with a sustainable social and economic impact. The success of these application scenarios will be assessed via extensive user experience assessment activities.

The following subsections describe in detail all the considered application scenarios.

4.1 Support to Financial Services

The Exus pilot focuses on the loan-origination use case. Bank clients requesting a banking product e.g. credit card or consumer loan, are required to present information including personal, professional, financial and other details to the banking institution, depending on which their request will be approved or rejected.

With the current infrastructure, anonymity is by default forfeited. Clients have to manually collect all the necessary documents and, on top of that, they reveal unnecessary personal details because the submitted documents include them by design. Finally, extra verification is necessary in order to ensure the authenticity of the submitted documents, and additional time will be spent profiling and scoring the client. This time-consuming and paperwork-intensive process can be vastly improved using the ABAC architecture.

Every piece of information that the banking institution requests can be individually revealed and certified by the corresponding authority -*Identity provider* in ABAC terminology- given the user’s consent. The process can be automated since all data is electronically exchanged, and verification will be effected with the use of electronically signed credentials. Furthermore, clients requesting a loan may retain their anonymity if the banking institution decides that some non-revealing information from a reputable authority is sufficient to approve the client’s request, for example instead of a full name and address the client presents only a Citizen’s Identification Number from the Government Taxation Office.

4.1.1 Before ABAC

The current situation in the Greek Banking environment involves mostly manual procedures. The steps taken, starting from the request submission until the purchase of a banking product, may differ depending on the product value and the risk involved. Extensive screening and detailed verification of personal and financial data is applied for high risk products such as business loans, mortgages etc. Simpler procedures are followed for lower risk products like consumer loans and credit cards. In all cases however, a lot of paperwork is required with reference to the documents that must be submitted with the client request, and a considerable amount of time will be spent for the verification of the submitted data.

Although credit cards request-forms can be submitted by post, for all other products the client will have to eventually visit the bank.

A bank employee will create a new (potential) client record and receive the client’s request-form along with the necessary documents that list the client’s personal information (financial, social security, health insurance, etc.). The documents’ authenticity will have to be verified either by simple inspection or, depending on the value of the banking product, by direct contact of the originating authority.

In the case of collaterals such as real estate, external partners such as real estate professionals may be called upon by the bank to visit and inspect the property on location.

In order to complete the client’s risk profile, the banking institution will collect additional data from organizations such as the National Banking Information System. This information concerns existing loans that the client has taken and loan payments that are overdue.

Finally, the banking institution may also collect information regarding the credit history of the client, and then calculate the risk of the client not being able to make the payments for the requested bank product.

At the end of the process the client’s request will be approved or rejected based on the risk score and the customer’s profile according to the banking institution’s policy.

The “loan origination” use-case closely adheres to the “attribute based access control” paradigm, considering that the user’s “banking profile details” are the attributes based on which the banking institution will control the user’s access to the institution’s services i.e. credit card issuance, consumer loan etc. The ABAC architecture provides considerable improvements regarding automation, efficiency, security, and privacy which are issues causing significant concern especially in the context of today’s banking environment.

4.1.2 After ABAC

Once the ABAC architecture is adopted, it is expected that the certificate-issuing authorities as well as the banking institutions will deploy servers offering ABAC functionality.

4.1.2.1 Identity Providers

The certificate-issuing organizations are the Identity Providers. They store and certify aspects of a user’s identity such as full name, address, telephone number, social security status, health insurance status, annual income, financial obligations (outstanding balance), etc.

4.1.2.2 Online Services

The banking institution is the Online Service offering its products to potential customers.

4.1.2.3 Behavioral Authentication Authorities

Organizations such as mobile telephony providers can function as Behavioral Authentication Authorities (BAA) providing second level authentication at the request of an Online Service. A mobile telephony provider may compare a user’s current location with the recorded user’s usual whereabouts based on the cell-towers that the user’s mobile device usually connects to. If there is no match the authentication will fail.

4.1.2.4 LATCH

Users may also configure BAAs to lock the account that a user has with an Identity Provider. In case multiple consecutive behavioral authentication attempts fail, a BAA’s ABAC server will use the LATCH service to lock all the user’s accounts according to the user’s specifications.

Taking all the above into consideration, the financial scenario is shaped as follows.

4.1.2.5 The Financial scenario revisited

A user that wishes to have a credit card or consumer loan issued by a banking institution will navigate to the banking institution’s web portal using the web browser of a mobile device (smartphone, tablet) or desktop computer. The bank’s web portal is the front-end to its ABAC infrastructure.

The user will be requested to provide certain information that will need to be validated. Such information includes, full name, age, annual income, social security status, employment status etc.

The user already has electronic credentials for some of these details stored in the mobile device. The user uses biometric authentication (fingerprint, voice, etc.) to unlock these credentials and submit them to the bank’s web portal.

The user also uses the mobile device to visit the web portals of other authorities and have electronic credentials issued for any additional information that the bank’s portal requests. The user either has a separate user-login with each of these authorities or may use OpenID and OAuth authentication in order to be identified at the authority’s server and have the necessary credentials issued. All the credentials are electronically signed by the issuing organization in order to ensure their authenticity. The credentials only include/reveal the specific information that the banking institution requests and not the total of the user’s records maintained by the credential-issuing authority. The issued credentials are transferred to the user’s mobile device and the user submits them to the bank’s web portal.

The bank’s ABAC server may in addition request for second level behavioral authentication. The BAA’s, e.g. Mobile Telephony Provider, ABAC server will retrieve the records of the user’s registered behavior, e.g. mobile cell-towers that the user mobile phone usually connects to, as an indication of the user’s normal whereabouts, and calculate the probability that the current user’s behavior matches the user’s profile or not. If the authentication fails, the BAA may also lock the user’s accounts at other ID Providers.

The bank’s ABAC server will examine and verify the credentials, consider the BAA’s response, and apply the banking institution “access control” policy. This policy specifies the criteria that must be fulfilled in order to accept or reject a client’s request for a credit card or consumer loan. If all criteria are met the user will be issued the credit card or granted the consumer loan.

4.2 Campus Wi-Fi and Campus-Restricted Web Services

The CUT pilot focuses on the access of students, professors and guests to the Campus Wi-Fi and other Campus Web Services. The students and professors requesting access to one or more Campus resources (e.g. research network with the permission to print), are required to present information including personal (such as first and last name, age, street address, etc.) and educational (such as year

of study, scholarship, teaching years, etc.). The guests must be vouched by a registered user in order to get access to some resources.

With the current infrastructure, the users reveal their full profile to get access to some resources with a lot of unnecessary information. They use the traditional username/password scheme to authenticate which nowadays is considered an insecure way on nowadays. They also get access to various resources by simple access control policies.

The authentication and the authorization procedure can be vastly improved using the ABAC architecture. The pilot before and after ABAC and the advantages of using the ABAC architecture are presented below.

4.2.1 Before ABAC

When the students and professors register with their university, the university’s IT services create and store their full profile within their infrastructure.

A user that wishes to get access to the campus Wi-Fi and the other resources can navigate to the University’s web portal using the web browser of his/her mobile device.

Then, the user can choose a resource and the Authorization Server transfers his/her full profile to the Authentication Server.

Afterwards, the user uses his/her university username and password in order to authenticate.

The Authentication Server confirms the user’s full profile and the Authorization Server receives from the Authentication Server the confirmed full profile of the user.

With the confirmed full profile, the username and the requested resource, the Authorization Server grants access to that resource according to the access control policies which defined by the network administrators. These policies specify the criteria that must be fulfilled in order to accept or deny a user’s request. If all criteria are met the user can get access to the requested resource.

4.2.2 After ABAC

When the students and professors register with their university, the university’s IT services create and store some identity attributes (such as full name, title or department, etc.) within their infrastructure. Furthermore, credentials of the identity attributes are issued and stored on user’s device.

A user that wishes to get access to the campus Wi-Fi and the other resources (with some permission) can open the Campus Access mobile application, insert his/her user id and choose one of the two available options.

First Option:

The user can see a list of all the available resources. Then, the user asks for access to one or more resources and the Campus Access mobile application informs him/her for the identity attributes required for each resource.

Second Option:

The user can see a list of identity attributes. Then, the user selects the attributes that he/she wants to reveal and the Campus Access mobile application informs him/her about the resources to which access will be granted, according to the selected identity attributes.

Afterwards, the Authorization Server transfers the user's id and the required identity attributes to the Authentication Server. The Authentication Server maintains an attributes database which provides a mapping between the user's id and his/her respective confirmed identity attributes.

The user uses biometric authentication (such as fingerprint) in order to authenticate against his/her device. Then the user goes with his/her device to the registrar which performs the FIDO registration (private key generated on the device), keeps the public key on the local identity provider attributes database and maps attributes to user. By using this mapping the Authentication Server confirms the required user's identity attributes.

After that the Authorization Server receives from the Authentication Server the confirmed identity attributes and the user's id.

With the confirmed identity attributes, the level of assurance of these identity attributes, the user's id, the requested resources and the level of criticality of these resources, the Authorization Server grants access to those resources (with some permission) according to the attribute-based access control policies which defined by the network administrators. The ABAC policies specify the criteria that must be fulfilled in order to accept or deny a user's request. If all criteria are met the user can get access to the required resources.

4.2.2.1 Advantages of ABAC

1. The users have the opportunity to use a very useful mobile application to get access to various resources instead of the navigation to the University's web portal to find a resource.

2. The users do not have to validate their complete identity and their full profile in order to get access to a resource. This approach allows users to reveal only a set of required identity attributes and not to reveal irrelevant parts of their profile.

3. The users can use biometric authentication (such as fingerprint) instead of regular username/password scheme, increasing the security.

4. The Authorization Server grants access to the requested resources according to the complex attribute-based access control policies instead of the simple policies. The ABAC policies also take into account the level of assurance of the required identity attributes and the level of criticality of the requested resources

4.2.2.2 Attributes and Attribute Values

The tables below present a list of identity attributes with a valid attribute value for a professor and a student of the Campus Wi-Fi and Campus-Restricted Web services use case scenario.

4.2.2.2.1 Professor:

Attribute	Attribute Values
First Name	John
Last Name	Andreou

Father's Name	Andreas
Gender	Male
Birth date	1970-02-15
Age	46
Nationality	Cypriot
Religion	Christian Orthodox
Street Address	15, Athinon
Country	Cyprus
City	Nicosia
Postal Code	2710
E-mail	john.andreou@cut.ac.cy (university), jandreou@gmail.com (home)
Phone Number	99-965423(mobile), 22-346529(home), 25-233303 (work)
Title	Professor
Department	Electrical Engineer, Computer Engineer and Informatics
Start Date	2009-09-10
End Date	-
Year of Study	-
Semester	-
Teaching Years	7
Chosen Courses	-
Teaching Courses	Control Systems, Electronics II, VLSI Systems Design
Number of passed courses	-
Scholarship	-

4.2.2.2.2 Student:

Attribute	Attribute Values
First Name	Helen
Last Name	Panayi
Father's Name	Stelios
Gender	Female
Birthday	1994-02-15
Age	22
Cypriot	Greek
Religion	Christian Orthodox
Street Address	2, Machiton Eldyk
Country	Cyprus
City	Limassol
Postal Code	4651
E-mail	helen.panayi@cut.ac.cy (university), hpanayi@gmail.com (home)

Phone Number	96-972325(mobile), 25-712093(home)
Title	Student
Department	Multimedia and Graphic Arts
Start Date	2012-09-10
End Date	2016-05-10
Year of Study	4
Semester	7
Teaching Years	-
Chosen Courses	Virtual Reality, Web Design, Operating Systems, Informatics
Teaching Courses	-
Number of passed courses	23
Scholarship	Yes

4.2.2.3 Online Services

The table below presents a list of the Online Services of the Campus Wi-Fi and Campus-Restricted Web services use case scenario.

Resource ID	Resource (permission)
1	**Internet**
2	**Moodle (upload content – Virtual Reality Course)**
3	**Moodle (upload content – Machine Learning Course)**
4	**Moodle (view content – Operating Systems Course)**
5	**Moodle (read/post messages to the Control Systems forum)**
6	**Webmail**
7	**Communication and Internet Studies Lab network (print)**
8	**Electrical Engineer Lab network (print)**

9	**Research network (use BitTorrent)**
10	**Research network (print)**

4.2.2.4 Attribute-Based Access Control Policies

Some examples of the attribute-based access control policies of the Campus Wi-Fi and Campus-Restricted Web services use case scenario are presented below.

"If a user is a **professor",** he can get access to the **Internet**.

"If the user's **first name** is **John** and his **last name** is **Andreou",** he can get access to the **Internet**.

"If a user is a **professor**, belongs to the **Electrical Engineer, Computer Engineer and Informatics Department** and he is responsible for the **teaching** of the **Pattern Recognition course",** he can get access to the **moodle** with the permission to **upload content for the Pattern Recognition course**.

"If a user is a **student**, belongs to the **Communication and Internet Studies Department**, he is in the **seventh semester** of his study and he **chose the Web Design course",** he can get access to the **moodle** with the permission to **view the content of the Web Design course**.

"If a user is a **guest,** he can get access to the **moodle** with the permission to **read the messages on the Control Systems forum.**

"If a user is a **student** and he **has chosen** the **Communication Theory course",** he can get access to the **moodle** with the permission to **post messages on the Communication Theory forum**.

"If a user is a **student,** belongs to the **Multimedia and Graphics Arts Department,** the **number of courses who has passed** is **more than 20** and his **current year of study** is **less than 6 ",** he can get access to the **Webmail**.

"If the user's **first name** is **Maria,** her **last name** is **Christou**, she has a **scholarship** and she belongs to the **Electrical Engineer, Computer Engineer and Informatics Arts Department",** she can get access to the **Electrical Engineer Lab network** with the permission to **print**.

"If a user is a **professor** with **more than two teaching years** in the CUT university**",** he can get access to the **Research Network** with the permission to **use BitTorrent**.

"If a user is a **student** and he is in the **fourth year of his study",** he can get access to the **Research Network** with the permission to **use BitTorrent**.

4.3 Age Verification

Age verification is based on UPCOM's Age Gate product, an attribute-based solution which can verify if a user requesting access to an age-restricted online resource is above a certain age, without revealing any other personal data. The online resource could be an age-restricted web site (e.g. porn or violence related), specific content (e.g. an NC-17 movie) or a purchase (e.g. alcohol or tobacco). The providers of those resources don't need to know any personal information of the users other than their age.

The Age Gate product uses various alternative methods in order to verify the user's birthdate, such as trusted physical ID providers (government authorities, banks, universities, etc.) and electronic ID cards. It can then use ABAC in order to issue cryptographic credentials, including only the (verified) birthdate attribute, according to which the service provider can grant or deny access to specific resources, based on well-defined policies. Those credentials are issued to the user's device, either by the ID consolidator or directly by a trusted IDP. The user can also backup those credentials to the IDC, so that they can be recovered, e.g. in case of device loss.

During the age verification application scenario, the following roles are identified:

- The user (user device), who acts both as a recipient, requesting age-related cryptographic credentials, and as a prover, against the online service that provides the requested age-restricted resource.
- The online service, who acts as the verifier of the user's age.
- The ID Consolidator and the trusted physical ID Providers, who act as issuers, issuing age-related cryptographic credentials to the user's device.

Following is the course of actions for the Age Verification scenario:

5. The user requests access to an age-restricted online resource.
6. The provider's server requests from the Age Gate service the initiation of the age verification process.
7. The Age Gate creates and returns a QR code, which is displayed to the provider's website.
8. The user scans the displayed QR code using the Age Gate mobile app.
9. The Age Gate mobile app authenticates the user (e.g. using biometric authentication).
10. The ReCRED daemon, located at the Service Provider, verifies the user's age, carried by the cryptographic credentials provided by the user's device, and matches them against local policies in order to grant or deny access to the requested resource.

4.3.1 Before ABAC

Age verification is an extremely important, yet very difficult to solve problem, especially without sacrificing user privacy. Current approaches fall into three main categories:

1. The user has to demonstrate evidence of ownership of a document which proves that he is an adult, such as a credit card or a driver's license. However, there are major issues with this approach. First of all, sensitive information is inevitably revealed and personal data could also be disclosed. In addition, this approach can't guarantee that the user is the legitimate owner of the provided proof. For example, a child could use her parent's credit card or driver's license, in order to access age-restricted content. Last but not least, proper age verification goes beyond the proof of being an adult. For example, the minimum age for drinking is USA is 21. Therefore, even if the user is the legitimate owner of a credit card, doesn't necessarily means that he is also above 21 years old.

2. The user has to demonstrate evidence of ownership of a document that explicitly states her birthdate, such as an ID card or a passport. With this approach, the service providers can determine the exact age of their users, however the rest of the problems with the first approach also apply here. Especially when it comes to the user's privacy, such legal documents usually include even more personal data than a credit card.

3. The service provider includes an age disclaimer, with which the user must agree in order to gain access to an age-restricted resource. This is a widely used approach, which is mainly used as legal cover for the service providers, hardly preventing any minors from accessing age-restricted resources.

4.3.2 After ABAC

We strongly believe that ABAC, along with the acquisition of the user's physical identity, can offer a new approach to age verification, which provides a solution to all the problems related with the current approaches.

The exploitation of the ABAC architecture benefits all involved parties. More specifically:

- The users can access age-restricted resources without having to share with the service providers any personal or sensitive data other than the absolutely necessary (their birthdate).
- The service providers can verify the age of their visitors and grant them access to age-restricted resources, according to specific policies that they can easily create and manage.

Minors are protected against age-restricted content and/or products.

4.4 ISIC Student Discounts

The ISIC Student Discounts pilot focuses on the student authentication and offers use case. A merchant or other real world service provider wishes to target student customers for a discount on their service or product. ISIC currently acts as an intermediary between student and discount provider in order to ensure that the service provider delivers a discount to a verified student. This is achieved through verification of the student's identity and enrolment at the time of purchase of an ISIC card which is then valid for a period of 16 months.

With the current process there is no mobile solution for the verification of a student customer's identity and student status. Service providers are required to do a visual check of the student's ISIC card in order to evaluate if the student is eligible for a discount. The information on the ISIC card is limited and therefore a service provider cannot create and evaluate eligibility of a student customer for complex discounts with multiple attributes based solely on the ISIC card. Furthermore, neither the service providers nor ISIC provide students with an application where they can receive targeted offers with a greater incentive to purchase as both parties do not have access to the student's specific attributes such as preferences or interests.

The information required to support service providers in verifying a student as a valid ISIC card holder can be provided by a trusted Identity Provider (IDP) after user consent. It can then use ABAC in order to issue cryptographic credentials, including for example only the (verified) department of enrolment according to which the service provider can grant or deny access to specific discounts, based on well-

defined policies. Those credentials are issued to the user’s device, either by the ID consolidator or directly by a trusted IDP. This includes attributes that are not present on the ISIC card but are registered during ISIC registration. Alternatively, the student can gain access to specific incentivized targeted discounts by giving consent to service providers to access their interests and preferences when creating targeted discount campaigns.

4.4.1 Before ABAC

The current situation for service providers giving discounts to students with an ISIC card is that they are required to do a visual check of the ISIC card prior to redeeming the discount. Given the international nature of the ISIC association and respectively the variation in design and quality of ISIC cards from different countries, a visual check of an ISIC card offers only a limited degree of confidence that a discount is being delivered to a verified ISIC card holder.

Furthermore, service providers are limited in their ability to create targeted discount campaigns for ISIC students since there are insufficient attributes displayed on the ISIC card to evaluate eligibility. All discounts created for ISIC students are available and presented to all ISIC students even though they may not be relevant or appropriate. When a service provider does want to limit availability of a discount this is only possible through description of the offer and relies on the student to ignore discounts that are not relevant for them. When a service provider then wants to verify the student’s eligibility for a discount they are required to request alternative proof for a specific attribute. This can be an official ID for attributes such as age or nationality but in certain case attributes cannot be verified.

Lastly, when the student wants to redeem a discount in the current situation, he does not have the option to limit the personal information that the service provider receives either with the visual verification of the ISIC card or when the service provider uses the current web services provided by ISIC. The service provider is therefore able to see specific personal information about the student which are irrelevant to the discount which is being provided. This is especially prevalent when the student is required to use multiple identification documents to prove certain requested attributes.

4.4.2 After ABAC

With the introduction of the ABAC architecture, service providers will be able to reference different IDPs in order to verify requested attributes in order to provide access to specific discounts. The pilot will deliver a campaign manager application which can be used to create these discounts and limit access based on specific attributes. For the purpose of the pilot discounts will be provided using a mobile app for ISIC students. Students will be able to manage their preferences and consent from the mobile app. Discounts provided to the mobile app will be limited to pre-purchase coupons for the pilot but with a wider adoption of the ABAC architecture, it is expected that the service providers will integrate the campaign manager application with the back-end of their cash registers allowing for in-store redemption and payment of discounted services and products.

Furthermore, students will be assured that service providers are only provided with the specific attributes which are related to the service that they are purchasing and that they have insight into which attributes are being provided to service providers as well as being able to manage access which service providers have access to their personal information.

5 Privacy and Security Considerations

5.1 Attacks and Privacy Issues in ABE and ABAC

In this section we describe identified attacks and drawbacks from a privacy point of view for ABE and ABAC systems. In general, from our survey we have observed that although many ABE and ABAC schemes have been proposed in the literature, there still no scheme that can eliminate all security and privacy loopholes found in these systems.

5.2 Lack of Revocation

User revocation is a major issue in ABE systems, where each attribute is conceivably shared by multiple users. As quoted in [25]: “Revocation of any single user would affect others who share his attributes. Moreover, user revocation in attribute based systems may be flexible and occur in different granularities. That is, it may require to revoke either the entire user access privilege, or just partial access right of the user, i.e., a subset of his/her attributes. Existing CP-ABE schemes suggest associating expiration time attributes to user secret keys.” For the same reasons described above, revocation is also difficult to achieve in ABAC systems.

5.3 Key Abuse Attack for KP-ABE

This attack was presented in [29], where the key abuse attack is introduced. In the KP-ABE, a user secret key is defined over an access structure and does not have the one-to-one correspondence with any particular user. As a result, a paid user is able to “share” his secret key and abuse his access privilege without being identified. More seriously, malicious users may take this advantage to make profits by abusing the access privilege. We call this kind of misbehavior as key abuse attacks.

5.4 Key Escrow

Most of the existing ABE schemes are constructed on the architecture where a single trusted authority, or a key generation center has the power to generate the whole private keys of users with its master secret information. A major drawback with these schemes is known as a key escrow problem [30]. The key generation center could decrypt any kind of messages addressed to specific users by generating their private keys. This is not suitable for data sharing typical scenarios where the data owner would like to make their private data only accessible to designated users.

5.5 Attribute Hiding Attack in ABAC

ABAC policy may grand access to protected resources based on two complementary ways. At the first one the user should hold a set of attributes in order to gain access. At the second the user should not have the specific attributes (e.g., User should not be from Europe). In such a case ABAC is vulnerable to attribute hiding attacks [31]. In this attack, a malicious user hides or alternates some attribute values that he holds in order to get access to a resource that would be denied otherwise.

5.6 Reveal of Access Policy and Attributes to Untrusted Servers

In existing constructions of ABE, either the access policy or attributes should be attached in plaintext to the data ciphertext to facilitate user decryption. These plaintexts, particularly data access structures

in ABE, reveal the data owner’s access policies when disclosed to untrusted servers, and hence have privacy concerns.

5.7 Reveal of User's Identity in Multi Authority Scheme

This privacy issue of CP-ABE scheme has been described (and resolved) in [32]. As the authors quote: “In multi authority attribute-based encryption schemes, a user can acquire secret keys from multiple authorities with them knowing his/her attributes and furthermore, a central authority is required. Notably, a user's identity information can be extracted from his/her some sensitive attributes. Hence, existing ABE schemes cannot fully protect users' privacy as multiple authorities can collaborate to identify a user by collecting and analyzing his attributes. Also the central authority has the power to decrypt every ciphertext, which seems somehow contradictory to the original goal of distributing control over many potentially untrusted authorities”.

5.8 Mitigations to Enhance Security in ABE and ABAC systems

In the literature, many ABE schemes have been proposed that try to eliminate the above security and privacy issues and at the same time try not to affect the performance of ABE systems. For example, in [33] the authors try to create a revocation scheme for ABE. In particular, the authors can revoke one attribute of a user instead of all attributes issued to him and the user can complete decryption as long as the unrevoked attributes of the user satisfy the access structure. A comprehensive review of various ABE schemes that address the identified issues for ABE can be found in [34] and [35].

5.9 Privacy considerations of Idemix and U-Prove

In this section, we analyze the privacy features and possible drawbacks of Idemix and U-Prove anonymous credentials. Despite the fact that these two systems provide significant privacy enhancements, we have observed that the literature does not include many works regarding their privacy issues or potential drawbacks.

5.9.1 Threat Model

Privacy properties are defined with respect to assumptions about adversaries, which could be inside the system, such as adversarial identity providers, or they could be external parties with access to some or all of the information available in the system. Adversaries can try to disclose information about user attributes or past actions through any available means. Multiple adversaries can collude with each other (including collusion between adversarial identity providers and service providers), and they might exchange information or messages outside the scope of the defined protocols. A party A that trusts a party B will believe not only that B is trustworthy, but also that B takes measures to prevent adversaries from gaining access to privileged and sensitive information, including mechanisms to defend against code vulnerabilities. Also trust is not necessarily permanent. That is, party A might cease to trust B if evidence emerges that indicates B might be untrustworthy [36].

5.9.2 Comparison of Privacy features

Idemix features two kinds of credentials: Idemix pseudonyms and Idemix anonymous credentials. An Idemix pseudonym is used for two-party returning-user authentication. An Idemix anonymous credential, on the other hand, is used for third-party open-loop authentication, and provides full privacy, including unobservability, anonymity, selective disclosure (including the ability to prove that a numeric attribute is greater or less than a numeric constant without disclosing its value), issue-show

unlinkability, and strong multi-show unlinkability by the same party or different parties. On the other hand, U-Prove has U-Prove tokens, which are used for third-party open-loop authentication. They provide unobservability, anonymity, selective disclosure (but not the ability to prove that a numeric attribute is greater or less than a numeric constant without disclosing its value), and issue-show unlinkability. However, they do not provide multi-show unlinkability [37].

In U-Prove technology, the issuance of credentials is based on blind signatures. The issuer thus cannot link an issued credential to the issuing session. However, for proving validity or attributes of a credential, besides a zero-knowledge proof of knowledge of the recorded attributes, the issuer’s signature is revealed. Different usages of the same credential are thus linkable. In order to ensure unlinkability, a used credential would have to be reissued after every usage. On the other hand, Idemix technology does not require revealing the signature for proving properties. Possession of a signature is proved with a zero-knowledge proof. Therefore, multiple shows of the same credential can remain unlinkable. As issuance is not based on blind signatures, the issuer can link an issued credential to its issuance session. However, this does not immediately reveal the link with subsequent sessions where the credential is shown. An important feature of Idemix technology is that it also allows to prove only properties of recorded attributes, such as a range of an attribute value. Further, it is possible to prove that committed values or values enclosed in a verifiable encryption are credential attributes [38].

5.9.3 Revocation

As mentioned in [39], unlinkability makes revocation difficult. The ability to revoke credentials is usually taken for granted. In the case of privacy-friendly credentials, however, it is difficult to achieve. An ordinary CRL (Certificate Revocation List) cannot be used, since it would require some kind of credential identifier known to both the issuer and the relying parties, which would defeat unlinkability.

U-Prove credentials can be revoked by users because they do not have multi-show unlinkability, but cannot be revoked by issuers, because they have issue-show unlinkability. Idemix credentials, which have both multi-show unlinkability and issue-show unlinkability, are revocable neither by users nor by issuers. U-Prove credentials have a Token Identifier, which is a hash of the public key and the signature. Because U-Prove does not provide multi-show unlinkability, the Token Identifier, like the public key and the signature, is known to all the relying parties. The user agent could therefore revoke the credential by including the Token Identifier in a CRL. However, because U-Prove provides issue-show unlinkability, the credential issuer does not know the Token Identifier, nor the public key or the signature, and therefore cannot use it to revoke the credential.

Idemix has a credential update feature that can be used to extend the validity period of a credential that has expired. This facilitates the use of short-term credentials that may not need to be revoked. However, the credential-update feature can be used to implement credential revocation. Moreover, short term credentials are an alternative to revocation, but they have drawbacks:

- short them credentials are costly to implement for the issuer;
- they impose a logistic burden on the user agent;
- they may become unavailable if the issuer is down when the validity period needs to be extended;
- the user agent may be overwhelmed by the need to renew many credentials at once if it has not been operational for an extended period of time.

6 Conclusions

This is the first deliverable from the WP5 of the ReCRED project. It provides the description of the investigations on the state of the art of Attribute Based Access Control technologies, protocols and architectures that should be used as a reference for the future development of the ReCRED ABAC architecture.

Together with the overview on the state of the art, one of the major contributions reported in this deliverable is the preliminary integration and deployment of components for anonymous credentials in the ABAC architecture of ReCRED. Indeed, as a result of the investigation on the state of the art, technologies such as Idemix and U-Prove, as well as the ABC4Trust architecture and the FIWARE RESTful API have been identified as a reference for the design and the development of the ReCRED ABAC architecture.

Moreover, WP5 is going to follow the DCA paradigm integrating attribute-based access control procedures directly on the user’s device by exploiting a Trusted Execution Environment, either among the ones already shipped by device vendors, or through more open platforms that allow custom development.

The privacy and security section is considering the initial definition of threats for the ABAC procedures as well as specific threats and limitations of the ABAC components and related mitigation techniques.

The results reported in this document will be used as input both for the project’s pilots and for the future work on the development of the ReCRED architecture, as well as of the ABAC architecture itself.

7 References

[1] Department of Defense Trusted Computer System Evaluation Criteria, DoD 5200.28-STD, December, 1985.

[2] NIST Joint Task Force (2013). Security and privacy controls for federal information systems and organizations. NIST Special Publication 800, 53.

[3] Ferraiolo, David, Janet Cugini, and D. Richard Kuhn. "Role-based access control (RBAC): Features and motivations." Proceedings of 11th annual computer security application conference. 1995.

[4] Sandhu, R., Ferraiolo, D.F. and Kuhn, D.R. (July 2000). "The NIST Model for Role Based Access Control: Toward a Unified Standard" (PDF). 5th ACM Workshop Role-Based Access Control. pp. 47–63.

[5] Ahn, Gail-Joon, and Ravi Sandhu. "Role-based authorization constraints specification." ACM Transactions on Information and System Security (TISSEC) 3.4 (2000): 207-226.

[6] Camenisch, Jan. "Specification of the Identity Mixer Cryptographic Library, Version 2.3.1, December 7, 2010."

[7] U-Prove Cryptographic Specification V1.1 Revision 3 - Christian Paquin, Greg Zaverucha

[8] ABC4Trust - https://abc4trust.eu/

[9] FIWARE - https://www.fiware.org/

[10] Privacy Open RESTful API Specification https://forge.fiware.org/plugins/mediawiki/wiki/fiware/index.php/Privacy_Open_RESTful_API_Specification

[11] FIDO Alliance – http://fidoalliance.org

[12] OpenID – http://openid.net

[13] OpenAuth – http://oauth.net

[14] Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Advances in Cryptology – EUROCRYPT 2001, vol. 2045 of LNCS, pp. 93-118. Springer Verlag, 2001.

[15] Camenisch, Jan, and Van Herreweghen, Els. "Design and implementation of the idemix anonymous credential system." Proceedings of 9th ACM Conference on Computer and Communications Security. ACM Press, 2002

[16] Jan Camenisch and Anna Lysyanskaya. Efficient non-transferable anonymous multi-show credential system with optional anonymity revocation. In Birgit Pfitzmann, editor, Advances in Cryptology — EUROCRYPT 2001, volume 2045 of Lecture Notes in Computer Science, pages 93–118. Springer Verlag, 2001.

[17] Jan Camenisch and Anna Lysyanskaya. A signature scheme with efficient protocols. In Stelvio Cimato, Clemente Galdi, and Giuseppe Persiano, editors, Security in Communication Networks, Third International Conference, SCN 2002, volume 2576 of Lecture Notes in Computer Science, pages 268–289. Springer Verlag, 2003.

[18] Bethencourt, John, Amit Sahai, and Brent Waters. "Ciphertext-policy attribute-based encryption." Security and Privacy, 2007. SP'07. IEEE Symposium on. IEEE, 2007.

[19] Chase, Melissa. "Multi-authority attribute based encryption." Theory of cryptography. Springer Berlin Heidelberg, 2007. 515-534.

[20] Lewko, Allison, and Brent Waters. "Decentralizing attribute-based encryption." Advances in Cryptology–EUROCRYPT 2011. Springer Berlin Heidelberg, 2011. 568-588.

[21] P2ABC Download - https://github.com/p2abcengine/p2abcengine

[22] https://github.com/credentials/credentials_idemix

[23] De La Piedra, Antonio, Jaap-Henk Hoepman, and Pim Vullers. "Towards a full-featured implementation of attribute based credentials on smart cards." Cryptology and Network Security. Springer International Publishing, 2014. 270-289
[24] Pydemix: Idemix implementation in python - https://github.com/netgroup/pydemix
[25] Shucheng Yu, Cong Wang, Kui Ren, Wenjing Lou, “Attribute based data sharing with attribute revocation”, ASIACCS 2010: 261-270
[26] Android Open Source Project, "Android Keystore System," [Online]. Available: https://source.android.com/security/keystore/index.html. [Accessed 3-2016].
[27] Android Open Source Project, "KeyGenerator," [Online]. Available: http://developer.android.com/reference/javax/crypto/KeyGenerator.html. [Accessed 3-2016].
[28] Android Open Source Project, "Cipher," [Online]. Available: http://developer.android.com/reference/javax/crypto/Cipher.html. [Accessed 3-2016].
[29] Shucheng Yu, Kui Ren, Wenjing Lou, Jin Li, “Defending Against Key Abuse Attacks in KP-ABE Enabled Broadcast Systems”, IACR Cryptology ePrint Archive 2009: 295 (2009)
[30] Xing Zhang, Cancan Jin, Zilong Wen, Qingni Shen, Yuejian Fang, Zhonghai Wu, “Attribute-Based Encryption Without Key Escrow”, ICCCS 2015: 74-87
[31] Jason Crampton, Charles Morisset, “PTaCL: A Language for Attribute-Based Access Control in Open Systems”, POST 2012: 390-409
[32] Jinguang Han, Willy Susilo, Yi Mu, Jianying Zhou and Man Ho Au, “Improving Privacy andSecurity in Decentralized Ciphertext-Policy Attribute-based Encryption”, IEEE Transactions on Information Forensics and Security, Vol. 10, Issue 3, pp. 665 - 678, 2015
[33] Qiang Li, Dengguo Feng, Liwu Zhang, “An attribute based encryption scheme with fine-grained attribute revocation”, GLOBECOM 2012: 885-890
[34] Zhi Qiao, Shuwen Liang, Spencer Davis, Hai Jiang, “Survey of attribute based encryption”, SNPD 2014: 1-6
[35] Cheng-Chi Lee, Pei-Shan Chung, Min-Shiang Hwang, “A Survey on Attribute-based Encryption Schemes of Access Control in Cloud Environments”, I. J. Network Security 15(4): 231-240 (2013)
[36] Eleanor Birrell, Fred B. Schneider, “Federated Identity Management Systems: A Privacy-Based Characterization”, IEEE Security & Privacy 11(5): 36-48 (2013)
[37] Francisco Corella, Karen Lewison, “Privacy Postures of Authentication Technologies”, ID360 Conference on identity, 2013
[38] Pros and Cons of Idemix for NSTIC – Pomcor, [Online] https://pomcor.com/2011/10/10/pros-and-cons-of-idemix-for-nstic/ [Accessed 3-2016]
[39] Pros and Cons of U-Prove for NSTIC, [Online] https://pomcor.com/2011/10/04/pros-and-cons-of-u-prove-for-nstic/ [Accessed 3-2016]
[40] U-Prove WS-Trust Profile V1.0 - Christian Paquin
[41] H2.2-ABC4Trust Architecture for Developers - Patrik Bichsel, Jan Camenisch, Maria Dubovitskaya, Robert R. Enderlein, Ioannis Krontiris, Anja Lehmann, Gregory Neven, Janus Dam Nielsen, Christian Paquin, Franz-Stefan Preiss, Kai Rannenberg, Michael Stausholm, Harald Zwingelberg
[42] FIWARE specification mention WS-Trust 1.4 - http://docs.oasis-open.org/ws-sx/ws-trust/v1.4/ws-trust.html