Integrating OpenID with proxy re-encryption to enhance privacy in cloud-based identity services

David Nuñez, Isaac Agudo, and Javier Lopez
Network, Information and Computer Security Laboratory
Universidad de Málaga
Málaga, Spain
Email: {dnunez, isaac, jlm}@lcc.uma.es

***Abstract*—The inclusion of identity management in the cloud computing landscape represents a new business opportunity for providing what has been called Identity Management as a Service (IDaaS). Nevertheless, IDaaS introduces the same kind of problems regarding privacy and data confidentiality as other cloud services; on top of that, the nature of the outsourced information (users' identity) is critical. Traditionally, cloud services (including IDaaS) rely only on SLAs and security policies to protect the data, but these measures have proven insufficient in some cases; recent research has employed advanced cryptographic mechanisms as an additional safeguard. Apart from this, there are several identity management schemes that could be used for realizing IDaaS systems in the cloud; among them, OpenID has gained crescent popularity because of its open and decentralized nature, which makes it a prime candidate for this task. In this paper we demonstrate how a privacy-preserving IDaaS system can be implemented using OpenID Attribute Exchange and a proxy re-encryption scheme. Our prototype enables an identity provider to serve attributes to other parties without being able to read their values. This proposal constitutes a novel contribution to both privacy and identity management fields. Finally, we discuss the performance and economical viability of our proposal.**

***Keywords*-identity management; privacy; OpenID; proxy re-encryption; cryptography; cloud computing.**

I. Introduction

Identity management is one of the most common services deployed within companies and organizations because of its key role in access control, authorization and accountability processes. However, it introduces an overhead in cost and time, and in most cases it requires specific applications and personnel for managing, integrating and maintaining this service. In the same vein as for other kinds of services, the cloud offers an innovative opportunity of externalizing this workload; this is what has been called *Identity Management as a service* (*IDaaS* or *IDMaaS*) [1]. IDaaS allows companies and organizations to outsource the identity management service from their internal infrastructures and deploy it on the cloud provider; that is, it permits moving identity management from an *on-premise* delivery model to an *on-demand* model. Additionally, IDaaS opens up a new business opportunity for cloud providers and vendors, broadening their service offering.

However, at the same time, IDaaS introduces a variant of one of the classic problems of cloud computing: the loss of control over outsourced data, which in this case is information about users' identity. The fact that this kind of information is protected by specific regulations, such as the European Data Protection Directive in the case of the EU [2], demands a strong protection of its storage, processing and communication. Traditionally, cloud providers have tackled these problems defining service level agreements (SLAs) and internal security policies; however, these measures simply reduce this issue to a trust problem. Nothing actually prevents providers from breaking these agreements and policies; users simply *trust* them not do it. In other words, there is an important trust problem, inherent to cloud computing – users want to have access to services but, at the same time, are unwilling to provide their data to entities that they don't necessarily trust. For these reasons, it would be desirable to have at our disposal more advanced security mechanisms that enable users to benefit from cloud computing and still preserve their privacy and the control over their information.

In this paper we demonstrate how an IDaaS system that preserves users' data privacy can be implemented. Our prototype implementation uses OpenID Attribute Exchange as the underlying identity management protocol, and a proxy re-encryption scheme for implementing the cryptographic protection. Our approach enables an identity provider to serve attributes to other parties without being able to read their values, preserving in this way users' privacy with respect to the identity provider. To our knowledge, this is the first approach that proposes a functional identity management system where the identity provider is unable to access users' information, which makes it a relevant contribution to the areas of privacy and identity management.

The rest of this paper is organised as follows: In Section II, we discuss related research on identity and privacy management. In Section III, we describe the OpenID protocol, as well as its Attribute Exchange extension. In Section IV, we explain some proxy re-encryption schemes and in particular, the one that we use in our prototype. In Section V we explain our proposed model, as well as some details about its prototype implementation and its evaluation; additionally, an economic analysis of this proposal is given. Finally, Section VI concludes the paper and future work is outlined.

II. Related work

The problem of privacy in identity management is a widely studied subject. In particular, much work has been carried out regarding unlinkability of users with respect to the other entities involved in the identity management processes. For example, in [3], the authors present PseudoID, a model for private federated login that achieves unlinkability of users to visited sites. To this end, a blind signature service participates during the generation of an access token that is handed to the identity provider; this access token consists of a pseudonym and a secret value, that are both used to anonymously authenticate the user. Although this work presents an interesting contribution to privacy-enhanced identity providers, it is centered in the unlinkability aspects of the authentication of users. Moreover, this model is not suitable for maintaining users' information in the identity providers, since the providers are unable to correlate users to their pseudonyms.

With regard to the intersection of identity management, privacy and cloud computing, there has also been some research done. In [4], the authors propose SPICE, an identity management system for cloud environments whose main goal is to preserve users' privacy. SPICE satisfies a set of properties that the authors claim an identity management system in the cloud should fulfill, such as unlinkability and delegatable authentication. In order to accomplish this, SPICE uses a re-randomizable group signature scheme. However, the aim of SPICE is not the same as ours, since we are not tackling unlinkability, but data confidentiality. In [5], a privacy-preserving identity management system for cloud environments is presented; this system is based on zero-knowledge proofs that allow the user to prove the knowledge of a set of attributes without revealing their value. The problem of heterogeneity of attributes representation is also addressed in this work by using ontology mapping techniques. However, the authors don't tackle the privacy issues that are the concern of our work, since in their setting, identity providers store in clear the values of the attributes of the users.

In [6], the authors propose a solution based on the deployment of active bundles in the cloud provider. An active bundle is a mobile agent, in this case a virtual machine, which contains the identity information of the user and that is protected by cryptographic means. Every time an operation involves the use of identity information, the cloud provider interacts with an active bundle to retrieve this information. However, this approach seems to be impractical because of the large overhead that introduces the use of a large container for data, such as a VM. Moreover, the proposal does not detail any procedure to transport these active bundles to the cloud in an efficient manner. Another related approach, presented in [7], uses active bundles, predicate encryption and multiparty computation to fulfill the same goal; however, it suffers from the same problems as the previous work, and introduces new ones, such as the overhead produced by the use of multiparty computation.

Another proposal, based on the use of sticky policies and trusted computing, is presented in [8]. This work presents an interesting approach where information, along with a specific policy that should be enforced in order to disclose the data, is obfuscated before leaving users' domain. In this approach, a trusted authority is in charge of giving the receiver the means to de-obfuscate the information, after verifying that the receiver complies with its associated policy; trusted computing is used to ensure the integrity of both software and hardware environments of the receiver. However, this work is focused on the direct sharing of information, which makes it unusable in an identity management setting, where an identity provider is used as an intermediary and must somehow manage this information.

III. OpenID

OpenID is a decentralized model for identity management, which allows service providers to delegate the authentication of users to identity providers. In this model, the identity of a user is represented by a URI, called OpenID identifier. Hence, users don't need to create a separate account for each site; instead, they just have to use their OpenID identifier, and the authentication procedure will be conducted through the user's identity provider.

Together with the OpenID 2.0 specification [9], the OpenID Attribute Exchange extension (OpenID AX) [10] was defined. This extension enables the exchange of users' attributes within the OpenID protocol flow. Specifically, it defines a mechanism for fetching user attributes that permits the service provider to include a petition for a set of attributes into an authentication request, and the identity provider to respond with the corresponding attribute values together with the authentication response. In addition, OpenID AX also defines mechanisms for enabling the service provider to store attributes in the identity provider.

In the OpenID context, the identity provider is usually called *OpenID provider* (*OP*) and the service provider, *relying party* (*RP*); however, we will adhere to the common convention, and we will refer to them as identity provider (IdP) and service provider (SP), respectively.

Next, we will consider the way that OpenID defines the exchange of identity information. Although OpenID is mostly used just for authentication, the OpenID AX specification enables the identity provider to convey user attributes during the authentication phase. The main flow of OpenID is shown in Figure 1 and is detailed as follows:

1) The user requests access to a service or resource at the SP site. At this moment, we assume that the user is not authenticated.

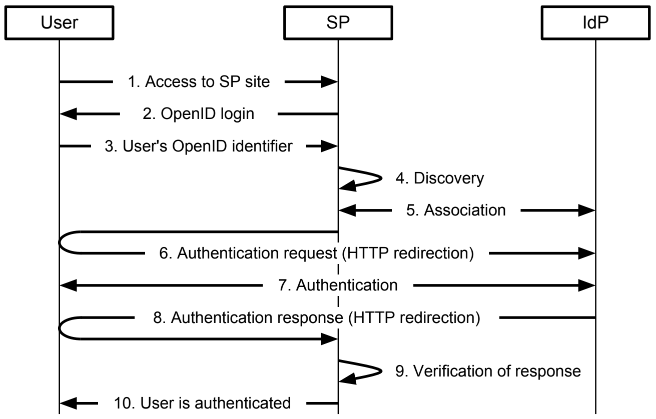
User
SP
IdP
1. Access to SP site
2. OpenID login
3. User's OpenID identifier
4. Discovery
5. Association
6. Authentication request (HTTP redirection)
7. Authentication
8. Authentication response (HTTP redirection)
9. Verification of response
10. User is authenticated

Figure 1. Sequence diagram of the OpenID Authentication protocol

2) The SP requires the authentication of the user and asks for his OpenID identifier. In order to do so, the SP shows the user a OpenID login page, where he can supply an OpenID identifier.
3) The user provides an OpenID identifier. He may have several identifiers, and he can choose which one to use. Additionally, OpenID 2.0 allows the user to simply provide the identifier of his identity provider, enhancing this way his privacy by reducing the chances of being traced through his identifier.
4) The SP performs a discovery process using the supplied identifier to locate the IdP of the user.
5) The SP and the IdP perform an association process, that is, they generate a shared secret through a Diffie-Hellman key exchange. This shared secret will be used to verify subsequent communications.
6) The SP constructs an authentication request and redirects the user to the IdP site through an HTTP redirection. We will assume that the Attribute Exchange extension is used, so the SP also includes a petition for a set of attributes into the authentication request.
7) The user gets authenticated by the IdP, for example, by providing his credentials. OpenID does not define a method of authentication, but password-based methods are the most common ones.
8) The IdP constructs an authentication response, which contains an assertion about the result of the authentication. In case the SP asks for attributes, the IdP also includes their values. Additionally, the IdP signs the request. The user is then redirected back to the SP site in order to continue with the authentication process.
9) The SP verifies the authentication response and reads the attribute values included within.
10) The user gets authenticated at the SP site and is able to access to the requested service.

OpenID is designed as an user-centric identity management system; the fact that the users are free to choose the identity provider, and even to establish their own provider, proves the user-centric nature of OpenID. However, there are some weak aspects of OpenID (some of them extensible to the rest of identity management systems) that are open to improvement:

- Identity information assurance, since the service providers need some mechanism to be confident that the provided identity information is true and valid.
- Trust, since the decentralized nature of OpenID enables to operate without requiring a pre-existing trust relationship between identity providers and other parties.
- Privacy, since the identity provider is in full control of users' identity information and is aware of users' access patterns.

In this paper we will tackle the privacy concerns respect to users' identity information; however, we acknowledge the importance of all aspects for realizing an integral solution.

IV. Proxy Re-Encryption schemes

From a high-level viewpoint, a proxy re-encryption scheme is an asymmetric encryption scheme that permits a proxy to transform ciphertexts under Alice's public key into ciphertexts under Bob's public key, as shown in Figure 2. In order to do this, the proxy is given a re-encryption key $r_{A \to B}$, which makes this process possible.

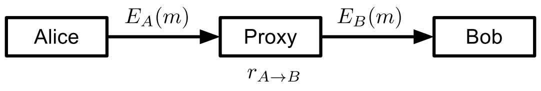

Figure 2. General proxy re-encryption sequence

The notion of proxy re-encryption was introduced in 1998 by Blaze *et al.* [11]; their proposal, which is usually referred as BBS scheme, is bidirectional (it is trivial to obtain $r_{B \to A}$ from $r_{A \to B}$) and multihop (the re-encryption process is transitive), but not resistant to collusions.

Ateniese, Fu, Green and Hohenberger proposed in [12] new proxy re-encryption schemes based on bilinear pairings. In particular, they provided an initial basic scheme, which is subsequently extended throughout the paper to support additional functionalities. Their scheme is unidirectional, unihop and resistant to collusions. Because of the simplicity of this scheme, we will use it in our implementation; we will explain it in more detail below.

Green and Ateniese proposed an identity-based proxy re-encryption scheme in [13]; however, this scheme is not resistant to collusions. An improved proposal that is secure against chosen-ciphertext attacks (CCA) is presented in [14], but again, it is not collusion-resistant.

In [15], Canetti and Hohenberger presented a CCA-secure bidirectional scheme; based on this security model, Libert and Vergnaud proposed in [16] an unidirectional scheme with chosen-ciphertext security in the standard model.

AFGH scheme. As aforementioned, Ateniese *et al.* defined in [12] an unidirectional, unihop and collusion-resistant proxy re-encryption scheme. This scheme is based in the ElGamal cryptosystem. Let $\mathbb{G}_1$ and $\mathbb{G}_2$ be two groups of prime order q, with a bilinear map $e : \mathbb{G}_1 \times \mathbb{G}_2 \rightarrow \mathbb{G}_2$; the global parameters are $g \in \mathbb{G}_1$ and $Z = e(g,g) \in \mathbb{G}_2$.

- *Key Generation* (KG): Alice selects a random integer $a \in \mathbb{Z}_q$ and generates her pair of secret and public keys, $s_A = a$ and $p_A = g^a$.
- *Re-Encryption Key Generation* (RKG): Alice takes Bob's public key p_B, and together with her secret key s_A, she computes the re-encryption key $r_{A\rightarrow B} = (p_B)^{1/s_A} = g^{b/a} \in \mathbb{G}_1$.
- *First-level Encryption* (E_1): Anyone is able to encrypt messages intended only for Alice using her public key p_A. To encrypt a message $m \in \mathbb{G}_2$, one selects a random integer $k \in \mathbb{Z}_q$, and computes $c = (e(p_A, g^k), mZ^k) = (Z^{ak}, mZ^k) \in \mathbb{G}_2 \times \mathbb{G}_2$.
- *Second-level Encryption* (E_2): To create second-level ciphertexts, which are re-encryptable, one computes $c = (p_A^k, mZ^k) = (g^{ak}, mZ^k) \in \mathbb{G}_1 \times \mathbb{G}_2$.
- *Re-Encryption* (R): Anyone in possession of the re-encryption key $r_{A\rightarrow B}$ can transform a second-level ciphertexts for Alice, $c_A = (\alpha, \beta)$, into a first-level ciphertext for Bob, by computing $c_B = (e(\alpha, r_{A\rightarrow B}), \beta) = (Z^b k, mZ^k) \in \mathbb{G}_2 \times \mathbb{G}_2$.
- *First-level Decryption* (D_1): As in other asymmetric encryption schemes, Alice uses her secret key s_A to transform a ciphertext $c_A = (\alpha, \beta) \in \mathbb{G}_2 \times \mathbb{G}_2$ into the original message m. In order to do so, Alice computes $m = \dfrac{\beta}{\alpha^{1/s_A}} = \dfrac{\beta}{\alpha^{1/a}}$.
- *Second-level Decryption* (D_2): For decrypting a ciphertext $c_A = (\alpha, \beta) \in \mathbb{G}_1 \times \mathbb{G}_2$, Alices computes $m = \dfrac{\beta}{e(\alpha, g)^{1/s_A}} = \dfrac{\beta}{e(\alpha, g)^{1/a}}$.

This scheme uses two different ciphertext spaces; first-level ciphertexts are intended for non-delegatable messages, whereas second-level ciphertexts can be transformed into first-level ones through re-encryption.

The AFGH scheme has the following properties:

- *Unidirectional*: The re-encryption key $r_{A\rightarrow B}$ cannot be used to derive the reverse one $r_{B\rightarrow A}$. This property is useful in settings where the trust relationship betwen Alice and Bob is not symmetrical.
- *Resistant to collusions*: If the proxy and Bob collude, they are not able to extract the secret key of Alice; at most, they can compute the weak secret $g^{1/a}$, but this information does not represent any gain to them.
- *Unihop*: This scheme is not transitive with respect to re-encryption; the re-encryption process transforms a ciphertext from one space to another, so this process cannot be repeated.

V. Privacy-preserving IDaaS system

In this section we will describe the main contribution of this work, a privacy-preserving IDaaS system that is able to provide identity information to other parties without being able to access the data, preserving in this way users' privacy. Our system is based in the use of proxy re-encryption, and in particular, the AFGH scheme, presented in the previous section.

We will restrict our work in this paper to a *honest-but-curious* provider, which will behave correctly with respect to fulfillment of the agreed protocol, but has no hindrance to access users' data. Hence, we will assume that the identity provider may have some incentive to read users' data without their consent.

A high-level diagram of our proposal is shown in Figure 3; this diagram depicts the main flow of information in our system, where the user encrypts his attributes under his public key p_U and sends them to the identity provider in the cloud. The use of a proxy re-encryption scheme enables the identity provider to transform these ciphertexts into encrypted attributes under the public key of the service provider, p_{SP}; in order to do so, the identity provider needs a re-encryption key $r_{U\rightarrow SP}$ generated by the user. Our system enables users to retain the control over the information and prevents the identity provider from reading users' data. Hence, there are three main actors that interact in this model:

- *User*: The user puts his attributes in the identity provider, but encrypted with his public key p_U. Being a user-centric approach, we can assume that only the user uploads information; however, since his public key is used for encrypting, anyone in theory is able to do so. If we want to prevent this, the user can refrain from disclosing his public key, as in our proposal it is only used for encrypting the attribute values and no other entity needs it. The user is also responsible for generating the corresponding re-encryption key for the service providers, since his secret key s_U is needed during this process.
- *Identity provider*: In this case, the identity provider acts as a proxy entity, which transforms ciphertexts from some user to ciphertexts of some service provider. In order to do so, the proxy needs the proper re-encryption key $r_{U\rightarrow SP}$, which is provided by the user.
- *Service provider*: The service provider receives ciphered attributes that can be decrypted using his secret key s_{SP}. From his standpoint, the process is transparent; somewhat simplistically, we could say that there is no proxy re-encryption for him, as he only has to decrypt ciphertexts with his own secret key.

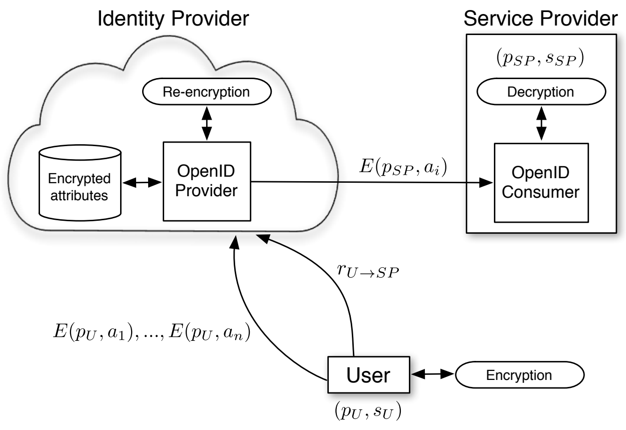

Figure 3. Integration of OpenID and proxy re-encryption

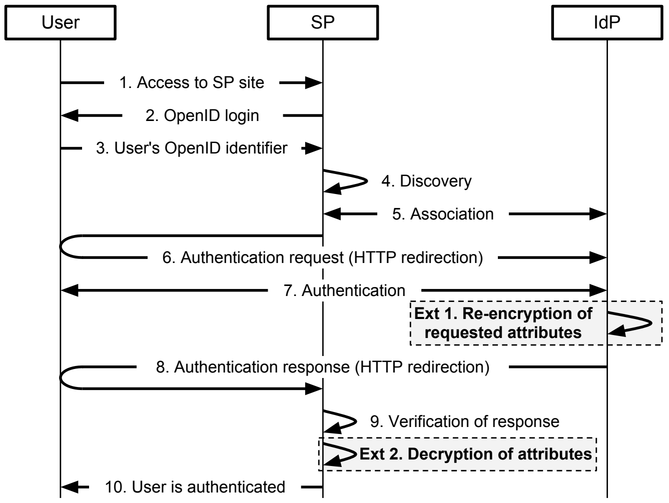
User
SP
IdP
1. Access to SP site
2. OpenID login
3. User's OpenID identifier
4. Discovery
5. Association
6. Authentication request (HTTP redirection)
7. Authentication
Ext 1. Re-encryption of requested attributes
8. Authentication response (HTTP redirection)
9. Verification of response
Ext 2. Decryption of attributes
10. User is authenticated

Figure 4. Sequence diagram of the modified OpenID protocol with proxy re-encryption

A. Instantiation with OpenID AX

In order to demonstrate the viability of our proposal, we decided to instantiate it using the OpenID Attribute Exchange 1.0 protocol as the underlying identity management scheme. Taking into account the steps described in Section III, the modified protocol is as follows:

When the user accesses the service provider, he is asked to authenticate himself; additionally, he may be asked to provide some attributes. The service provider constructs an OpenID authentication request, including also an Attribute Exchange extension header asking for a determinate set of attributes. This request is sent to the identity provider through a HTTP redirection on the user browser; the identity provider receives the request and, upon a successful authentication of the user, constructs an authentication response. This response will also include the values of the attributes requested by the service provider; in order to do so, the identity provider must use the appropriate re-encryption key (the one that corresponds to the specific user and service provider) to transform the attribute values encrypted for the user into ciphertexts for the service provider. Once the re-encryption is performed, the IdP sends the authentication response, including the attribute values encrypted under the public key of the service provider. The final step is for the service provider to decipher the attribute values using its secret key.

From a low-level standpoint, our proposed implementation respects almost entirely the regular OpenID AX protocol flow since it is extended in just two points, as shown in Figure 4:

- *Extension 1*: when the identity provider constructs the response, since it has to perform the re-encryption of the attribute values.
- *Extension 2*: when the service provider receives the encrypted values in the response, since it has to decrypt them.

B. Design issues

Global parameters. Since all the cryptographic computations must be performed under the same system parameters, for our prototype we have simply assumed that the global parameters are previously known by every party; however, in a real setting these parameters should be somehow published or distributed to the rest of the parties.

One option is that there is a trusted entity that generates and publishes the parameters, so this information is public to the rest of entities that could then compute their public keys; in this case, these parameters could be even signed by this entity for verification purposes. Another alternative is that these parameters are particular to each identity provider; however, this implies that each user and service provider must have a different pair of keys for each identity provider, which could be not feasible.

Encryption of user's attributes. A fundamental principle behind our proposal is that the user must be the responsible for encrypting his personal information and managing his keys, since only this way we can fulfill the primary goal of keeping the control of his data. However, this is not an easy task in a real setting, where the user normally interacts through a web browser. The browser must then be able to perform the encryption process, which can involve complex operations depending on the proxy re-encryption scheme used. Nevertheless, the support for cryptographic libraries in browsers is currently very limited; a more powerful cryptographic support for the browser could be very useful to perform this operation on the fly, as discussed in [3].

A first alternative to tackle this problem is using client-side scripting technologies; for example, the identity provider can include a piece of code in the page used to upload user information, so the encryption process takes place in the user side. However, this option introduces the problem of trusting such code, since it would be offered by the identity provider; this issue could be addressed through code signing by a trusted entity, but this introduces more complexity into the system. Another option is to use a browser plugin, or directly, a specific application to perform this operation; however, this alternative complicates the adoption of this proposal, since it obliges the user to install additional software. The suitability of these alternatives depends on the requirements of the specific use case.

Another important issue is that we just encrypt the value of each attribute, so its name is in clear; this approach makes our model easily integrable with existing directory services. In addition, the process of uploading user attributes to the identity provider is outside the scope of the OpenID AX specification; for all these reasons, we decided in our prototype to upload the encrypted attributes off-line. It is also worth mentioning that we do not directly encrypt attributes with the proxy re-encryption scheme; instead we use an hybrid approach. More details are given in Section V-C.

The encryption of user attributes is an important point of our system from the standpoint of security. Nowadays, the possibility of malware on users' devices and platforms is constantly increasing; for this reason, the identity provider needs to be sure of the integrity of the user's platform, and specifically, the application that encrypts the attributes. In order to tackle this matter, we could use remote attestation techniques [17]; for instance, in [18], Trusted Computing technology is used to enhance OpenID by providing both authentication and remote attestation of user's platform, enabling this way a link of trust between the identity provider and the user's platform. However, although this issue is relevant to the security of our solution, we consider it out of the scope of this paper.

Re-encryption keys. The re-encryption key, apart from making possible the re-encryption of ciphertexts, can also be seen as an access token, as it is created by the user when he wants to grant access to his attributes to some service provider. In the same manner, the re-encryption key can also be removed from the identity provider for revoking the access to user's attributes; we can trust that it will remove the re-encryption key when asked, as we assume that the identity provider is a honest-but-curious entity. Ideally, temporary re-encryption keys would be used, so access would be valid only for a specific period of time. To date, we have not seen any proxy re-encryption scheme that deals with re-encryption keys that are valid for a particular time period. In [12], a temporary proxy re-encryption scheme is proposed, which restricts both the re-encryption key generation and encryption processes to the same time period; however, this is not what we desire, since in our model the re-encryption key generation process will probably happen in a later time period. We leave as an open problem the design of a proxy re-encryption scheme with support for temporary or expirable re-encryption keys; our proposal would certainly benefit from such scheme.

In our prototype, we opted to create the re-encryption keys beforehand; however, in a real setting this process would be ideally conducted when a service provider asks for attributes for the first time. Again, we face the same problem we have with encryption, since complex operations must be performed on the user side; thus, the same alternatives can be applied here.

Recall that the public key of the recipient is needed during the re-encryption key generation. Thus, it would be advisable to use some standard mechanism of discovery and publication of public keys within the OpenID protocol, such as the one proposed in the draft specification OpenID Service Key Discovery 1.0 [19]; that way, the service provider can publish his public key without requiring any mechanism external to OpenID.

C. *Implementation and performance evaluation*

In order to make a quantitative evaluation of our proposal, we developed a prototype implementation [20], using Java as coding language. For the implementation of both identity and service providers we have used the OpenID4Java library [21]. We chose this library because it covers most of the OpenID specifications, and in particular, the Attribute Exchange extension. Our execution environment was an Apple Macbook Pro laptop, with a 2.66 GHz Intel Core 2 Duo processor and 4 GB of RAM, running Mac OS X 10.6.8.

With respect to the proxy re-encryption scheme, we have implemented it using the jPBC library [22], a pairing-based cryptography library for Java. As for the cryptographic details, we used a Type A elliptic curve with a 256-bit group order and 1536 bits for the field size, which achieves a 128-bit security level. Additionally, we have made extensive use of exponentiation preprocessing of frequently-used elements for efficiency reasons.

As previously mentioned, we have used a hybrid approach for encryption, as recommended when using asymmetric encryption algorithms. Instead of directly encrypting the attributes using the proxy re-encryption scheme, we are encrypting a fresh 128-bit key for each attribute value, which is then passed to a symmetric encryption algorithm (AES-128) for encrypting the attribute value. This way, the proxy encryption primitive is only used to cipher a short input (a 128-bits key), whilst the symmetric algorithm performs the bulk of the work. This approach is used not only for efficiency, but for input length reasons, since attribute values have a wide range of possible lengths.

The median execution times for the main operations, as well as an estimate of the number of cycles, are presented in Table I. Note that although the re-encryption operation is slower than encryption and decryption, in a real setting it would be performed by a cloud provider that presumably

Table I
PERFORMANCE RESULTS FOR THE MAIN OPERATIONS

Operation	Time (ms)	Cycles
Generation of global parameters	7279.98	1.94E+10
Generation of a secret key	0.01	1.86E+04
Generation of a public key	20.05	5.33E+07
Generation of re-encryption key	139.66	3.72E+08
Encryption	23.31	6.20E+07
Re-encryption	90.09	2.40E+08
Decryption	14.28	3.80E+07

Table II
COSTS FOR THE MAIN OPERATIONS

Operation	Cost per operation (in picocents)	Operations per cent
Encryption	4.34E+08	2304
Re-encryption	4.79E+08	2087
Decryption	5.70E+08	1755

counts on a much more powerful computing environment. In a realistic scenario, re-encryption and decryption are the most frequent operations, since they constitute the equivalent to a 'read' operation. The number of cycles gives an approximation of the workload involved in the main operations, independently of the frequency of the CPU; this metric will be of use in the next subsection. In order to estimate the number of cycles per operation, we simply multiply the time of execution by the frequency of the CPU of the experimental machine; however, note that this approach give us an overestimation of the number of cycles, as it is difficult to isolate the execution from other environmental interferences such as multitasking and I/O operations, therefore the actual figures are probably lower.

D. *Assessment from an economic perspective*

Cloud computing is currently a hot research topic, and as such, it is subject of a large quantity of proposals for enhancing its security, reliability and functionality. However, in current literature there are almost no critical analyses about the economic impact that can arise from the implementation of such proposals. In particular, this is the case of cryptography-based proposals, as they often imply an intensive use of computation and communication resources. Hence, in this paper we will try to give a rough estimation on the costs caused by our system.

As a basis for our analysis, we will consider the work presented in [23], where the authors explore the economic dimension of the outsourcing of computation and storage to the cloud, and in particular, of common cryptographic operations, such as AES and RSA. There exist diverse factors that influence the costs in cloud environments, being hardware, energy and personnel, the most significant ones; there are, however, other factors that have an indirect impact on the costs, such as taxes, credits, and insurances. One of the most valuable contributions of their work is the quantification of the costs for several aspects of the cloud. Taking the *picocent* (equivalent to 10^{-12} USD cents) as unit of cost, the authors break down the expenses derived from computation, storage and network service: in the case of computation, the prices range from 0.93 and 2.36 picocents per CPU cycle, depending on the cloud provider; for networking, costs may vary from 800 to 6 000 picocents per bit; and for storage, less than 100 picocents per bit per year. It is important to note that these costs are probably lower at this moment, since this work was presented in 2010.

For our analysis we will first consider the computation costs and assume, on the basis of the figures presented in [23], that the costs incurred by the cloud provider are 2 picocents per CPU cycle, 7 picocents for a home client and 15 picocents for a medium size company. Since the computation in our system is distributed – encryption is done on the user side, re-encryption at the cloud provider and decryption is performed by the service provider – the costs will also be distributed. It is important to mention that we don't consider the costs incurred by the OpenID protocol itself, as we are only evaluating the economic impact of the additional overhead that results from the protection of users' information through proxy re-encryption.

As for the communication costs, even assuming a cost of 5 000 picocents per transferred bit and that each attribute value has a length of 1 KB, which are both fairly high estimations, we are able to transmit almost 20 000 ciphered attributes per cent. Thus, communication costs are not a relevant factor to our analysis, since the information represented in form of attributes is usually short (less than 1 KB). Storage costs are negligible for the same reason, so we will not take them in consideration either.

Table II presents the results of this analysis. Apart from the cost in picocents of the main operations, we present a more tractable figure, the number of operations that can be performed per cent. As an illustrative example of the viability of our proposal, consider the border case of an IDaaS system that receives a million of attribute requests per day; that is, this cloud provider must perform a million re-encryptions per day. This represents an expense of 1749.29 USD on a yearly basis, since the cost for a re-encryption is 4.79 E+08 picocents; this additional cost would be passed on to the company that is externalizing their identity management service. We think these figures are reasonable for an average-sized company, but ultimately it would depend on the savings from outsourcing their identity services to the cloud. The costs that the company could incur in case of some disclosure or security breach are difficult to estimate, but at the very least it would have a negative impact with regard to reputation and loss of customers, and even fines or legal sanctions.

VI. Conclusions

Identity management as a Service (IDaaS) is a recent cloud computing paradigm that allows companies and organizations to benefit from outsourcing one of the most common and needed services. The reduction of costs and time-consuming tasks associated to managing identity services are the main reasons motivating this externalization. However, cloud computing has raised much concern with regard to the inversion of the control of the data. Our proposal provides an identity management service that guarantees user's privacy and control even when data storage and processing is performed by untrusted clouds.

The main contribution of this work is a novel and practical privacy-preserving IDaaS system based in OpenID Attribute Exchange and a proxy re-encryption scheme, proposed by Ateniese *et al.* in [12]. In our proposal, the identity provider is able to transform encrypted attributes from the user into ciphertexts for the service provider, without the identity provider being able to read user's attributes during this process. Details of the implementation and a performance evaluation are also given. In addition, we provide an economic analysis of our proposal, which proves quantitatively its viability; this kind of analysis is seldom presented in other works, so we consider it as a valuable contribution.

In this paper, we intended to tackle user's privacy issues with respect to identity provider; however, there are still other aspects regarding user's identity information such as unforgeability and assurance that are not handled by OpenID, and thus, not covered in our proposal; addressing these problems is left as future work. As other future work, we plan to test our implementation in a real cloud setting, such as Amazon EC2 or Google AppEngine; in addition, more recent proxy re-encryption schemes could be used in order to provide more efficiency or security. A further possibility to extend our work is to develop another prototype implementation for a different identity management protocol, such as SAML or Shibboleth.

Acknowledgments

The work in this paper was partly supported by the Spanish Ministry of Economy and Competitiveness through the SPRINT project (TIN2009-09237) and by the Junta de Andalucia through the projects FISICCO (P11-TIC-07223) and PISCIS (P10-TIC-06334). The first author has been funded by a FPI fellowship from the Junta de Andalucia.

References

[1] "Security guidance for critical areas of focus in cloud computing, version 3.0," Cloud Security Alliance, Tech. Rep., 2011.

[2] "Council Directive 95/46/EC: On the protection of individuals with regard to the processing of personal data and on the free movement of such data," 1995.

[3] A. Dey and S. Weis, "PseudoID: Enhancing privacy in federated login," in *Hot Topics in Privacy Enhancing Technologies*, 2010, pp. 95–107.

[4] S. Chow *et al.*, "SPICE–simple privacy-preserving identity-management for cloud environment," in *Applied Cryptography and Network Security*. Springer, 2012, pp. 526–543.

[5] E. Bertino *et al.*, "Privacy-preserving digital identity management for cloud computing," *Bulletin of the IEEE Computer Society Technical Committee on Data Engineering*, vol. 32, no. 1, pp. 21–27, 2009.

[6] P. Angin *et al.*, "An entity-centric approach for privacy and identity management in cloud computing," in *29th IEEE Symp. on Reliable Distributed Systems*, 2010, pp. 177–183.

[7] R. Ranchal *et al.*, "Protection of identity information in cloud computing without trusted third party," in *29th IEEE Symp. on Reliable Distributed Systems*, 2010, pp. 368–372.

[8] M. Casassa Mont, S. Pearson, and P. Bramhall, "Towards accountable management of identity and privacy: Sticky policies and enforceable tracing services," in *Proc. 14th International Workshop on Database and Expert Systems Applications*. IEEE, 2003, pp. 377–382.

[9] "OpenID Authentication 2.0," http://openid.net/specs/openid-authentication-2_0.html.

[10] D. Hardt, J. Bufu, and J. Hoyt, "OpenID Attribute Exchange 1.0," http://openid.net/specs/openid-attribute-exchange-1_0.html.

[11] M. Blaze, G. Bleumer, and M. Strauss, "Divertible protocols and atomic proxy cryptography," *Advances in Cryptology—EUROCRYPT'98*, pp. 127–144, 1998.

[12] G. Ateniese *et al.*, "Improved proxy re-encryption schemes with applications to secure distributed storage," in *Proceedings of the 12th Annual Network and Distributed System Security Symposium*, 2005, pp. 29–44.

[13] M. Green and G. Ateniese, "Identity-based proxy re-encryption," in *Applied Cryptography and Network Security*. Springer, 2007, pp. 288–306.

[14] C. Chu and W. Tzeng, "Identity-based proxy re-encryption without random oracles," *Information Security*, pp. 189–202, 2007.

[15] R. Canetti and S. Hohenberger, "Chosen-ciphertext secure proxy re-encryption," in *Proceedings of the 14th ACM conference on Computer and communications security*. ACM, 2007, pp. 185–194.

[16] B. Libert and D. Vergnaud, "Unidirectional chosen-ciphertext secure proxy re-encryption," *Information Theory, IEEE Transactions on*, vol. 57, no. 3, pp. 1786–1802, 2011.

[17] G. Coker *et al.*, "Principles of remote attestation," *International Journal of Information Security*, vol. 10, no. 2, pp. 63–81, 2011.

[18] A. Leicher *et al.*, "Trusted computing enhanced openid," in *Internet Technology and Secured Transactions (ICITST), 2010 International Conference for*. IEEE, 2010, pp. 1–8.

[19] G. Monroe and C. Howells, "OpenID Service Key Discovery 1.0 - Draft 01," http://openid.net/specs/openid-service-key-discovery-1_0-01.html.

[20] "Proxy re-encryption and OpenID," http://www.nics.uma.es/dnunez/proxy-openid.

[21] "OpenID4Java – OpenID 2.0 Java libraries," http://code.google.com/p/openid4java/.

[22] A. D. Caro, "Java Pairing Based Cryptography Library (jPBC)," http://gas.dia.unisa.it/projects/jpbc/.

[23] Y. Chen and R. Sion, "On securing untrusted clouds with cryptography," in *Proc. 9th annual ACM workshop on Privacy in the electronic society*. ACM, 2010, pp. 109–114.