Personal Identification in the Web Using Electronic Identity Cards and a Personal Identity Provider

André Zúquete, Hélder Gomes, Cláudio Teixeira

Personal Identification in the Web Using Electronic Identity Cards and a Personal Identity Provider

André Zúquete[1], Hélder Gomes[2], Cláudio Teixeira[1]

[1] DETI, IEETA, University of Aveiro, Portugal
[2] ESTGA, IEETA, University of Aveiro, Portugal

Abstract. This paper presents a new paradigm for implementing the authentication of individuals within Web sessions. Nowadays many countries have deployed electronic identity cards (eID tokens) for their citizens' personal identification, but these are not yet well integrated with the authentication of people in Web sessions. We used the concept of Personal Identity Provider (PIdP) to replace (or complement) the role ordinarily given to institutional Identity Providers (IdPs), which are trusted third parties to which service providers delegate the identification and the authentication of their clients. By running locally on a citizen's computer, the PIdP paradigm is well suited to assist his/her eID-based authentication. In this paper we describe an eID-based authentication protocol handled by a PIdP, its implementation and its integration in a production scenario (a campus-wide, Shibboleth IdP-based authentication infrastructure used in University of Aveiro).

1 Introduction

Within a Web context, user authentication is critical, especially when accessing personalized services/information. When the Web first appeared, people had to setup accounts on each and every service requiring client authentication. Then the Web evolved towards a more centralized and less annoying client authentication paradigm, using Identity Providers (IdPs). An IdP can centralize the authentication of a set of persons and provide identity attributes of authenticated individuals to authorised services. Authentication paradigms such as SAML 2.0 Web browser based SSO profile [1] explore this IdP-based authentication and identification services.

In this paper we build upon this IdP paradigm but we introduce a novelty: the authentication is performed by personal cryptographic tokens (smartcard-based identity cards, or eIDs) and the IdP is deployed and managed by the persons being authenticated, instead of some trusted third party. The benefits are the following: (i) people don't have to use more than their own eID to get authenticated, (ii) neither people nor services need to trust on third-party IdP services, other than the eID providers, and (iii) we can achieve a higher control over the authentication with an eID when comparing with the one performed by Web "transport" layers (e.g. by HTTPS [2]).

1.1 Motivation

According to [3], there are three types of eID solutions: password-based systems, Public Key Infrastructures (PKI) and attribute-based credentials. Our work is targeted to the PKI type, which is actually available in 26 European countries.

In eID solutions based on PKI, part of the citizens' identity attributes (name, civil identity number, birthday, sex, etc.) are sealed inside public key certificates, which also contain the public key corresponding to the private one with which signatures can be made. Signatures with personal private keys can only be made with the help of the eID. For this purpose, the eID is frequently a tamperproof smartcard with cryptographic capabilities and simultaneously a digital container for personal attributes. Private keys are maintained secret inside the eID smartcard, and their use in signature operations needs to be authorized with a personal 4-digit PIN. The physical protection provided by the smartcard, together with the PIN knowledge, provides a multi-factor authentication of the eID owner.

Most Public Administration Services, as well as many other public or private services (utility services, banks, insurance companies, etc.), usually require citizens to provide face-to-face their real identity to become clients. An eID simplifies the access of a citizen to all these services, since a citizen can use it to create an account through the Internet without having to show up somewhere for a face-to-face identification, and can use the same eID for accessing afterwards that account. Moreover, the exact same eID can be used to access many of such services services.

The trustworthy identity attributes that an eID can provide to online services depends on each eID. For eID solutions based on a PKI, these attributes must be part of a public key certificate associated with the eID (more precisely, associated to a private key stored in the eID).

Public key certificates (PKC) corresponding to eID private keys are signed documents that can be checked by anyone using certificate hierarchies provided by national PKIs. Any service should be able to check the validity and correctness of a personal PKC with the help of PKI validation services (e.g., CRL [4] distribution points). Thus, a person can get identified and authenticated by signing some challenge provided by the target service (the authenticator) with one of the person's private keys and presenting, together with the signed data, his/her corresponding PKC. The PKC enables the service to both (i) authenticate the person and (ii) get his/her identity attributes present in the PKC.

1.2 Problem

Currently, most browsers are able to explore cryptographic tokens (namely, eIDs) to authenticate the client side of an HTTPS session with an asymmetric key pair. However, server-side Web applications are designed in a way that make them independent from the so-called *transport layer*, which can be either HTTP or HTTPS. In other words, the exploitation of HTTPS instead of ordinary HTTP is a concern of the HTTP server, and not a concern of the applicational logic. Consequently, applicational sessions are completely independent from HTTPS

sessions, thus although the former are able to receive information about the authentication and identification of clients, they cannot completely rely on it, simply because they are unable to control it.

Since application-layer sessions cannot use the client authentication provided by the transport layer, they need to manage their own authentication policies and mechanisms. Thus, the application session control manager should directly interact with clients to authenticate them, possibly with an eID. But how can this be done without changing client applications, namely the Web browsers?

1.3 Contribution

For solving this problem, we propose to adopt the current IdP-based paradigms for authenticating people, namely SAML 2.0 Web browser based SSO profile [1], which is based on HTTP redirections, but using a personal IdP (**PIdP**) for making the proper interface with eIDs. The PIdP does not need to be a component trusted by the authenticating service; that will solely trust on the security robustness of eIDs and on the correctness of the eIDs PKI. From the service's perspective, the PIdP will act as a functional extension of a browser (it may inclusively be implemented as a plugin).

We designed a new authentication protocol involving a service provider (authenticating part), a PIdP and an eID, namely the Portuguese one. This protocol is robust against the theft of identity proofs, and protects also the identity of the authenticated person from networks eavesdroppers. A prototype of the PIdP was implemented in Java, as well as a set of Servlets for handling the authentication on the server side, and both components were integrated and temporarily deployed for demonstration purposes with the campus-wide, centralized and IdP-based authentication system used in the University of Aveiro.

2 Related work

One methodology currently in use for exploring the authentication with eID cards in Web iterations is based on the download and execution, just in time, of Java applets that interact with the eID. This approach is being explored by the majority of the Portuguese sites that allow Portuguese citizens to use their eID to authenticate themselves. However, this is not a good solution for several reasons. First, because browsers usually present a danger warning regarding the execution of such applets, because they can interact extensively with the user machine. Second, because users need to trust code that can be malicious (the service requesting eID authentication can be managed by attackers).

This approach is also the one followed by the e-Contract company[3] for the Belgian eID and by the eID Identity Provider project[4]. In this project they developed an IdP that interacts closely with an eID Applet that interacts with

[3] `http://www.e-contract.be`
[4] `http://code.google.com/p/eid-idp`

the eID. Our goal was different, we do not want to provide a completely new, fully-fledged IdP capable of interacting with eIDs. Instead, we provide a basic identification and authentication protocol, involving installed PIdPs for dealing with eIDs (see Section 4), which can then be used by existing IdP's.

A different approach [5], also put forward for the Belgian eID, consists in using a different HTTP protocol anchor (`auth`). In a browser, this anchor could be associated to a client authentication application, which could use the URI host and path following the anchor to contact an authentication server, or some IdP. The authentication protocol, which may, or may not, involve an eID, includes some HTTP client-server state information (e.g. an SSL session identifier) to establish the link between the authentication proofs and the relationship with existing (SSL) sessions. This approach involves changing all browsers to tackle this new anchor, and entangles the authentication proof with transport information (the SSL session identifier), which is not advised to keep application-level session management separated from secure transport management.

3 Architectural overview

In Web interactions, the most frequent way to separate the authentication and identification of people (service clients) from the actual service provisioning is based on the SAML 2.0 Web browser based SSO profile, presented in Figure 1 (top-left). When a browser first accesses a Service Provider (SP), the SP redirects the client to an IdP (for this reason, the service is also called Relying Party). The IdP authenticates the browser user, using some Web interface, and redirects the browser to an identity management module of the Relying Party. In this redirection, the IdP sends an assertion containing a set of identity attributes associated with the authenticated user. Upon validating the IdP assertion, the identity management module will select an existing profile matching the provided attributes and will thereafter use it in the subsequent interaction with the client (using a cookie returned after a profile match). Different services using the same IdP can get different sets of identity attributes for the same person.

Our simplest eID-based identification architecture consists in replacing the IdP by a PIdP located in the client host (top-right diagram of Figure 1). The PIdP conducts the eID-based user authentication and provides a user identification assertion, in the form of a datum signed by the eID, to the Relying Party's identity module. The user identity attributes are the identity attributes contained in the PKC that goes along with the signature. Upon validating the user signature, the identity management module will select an existing profile matching the user's PKC identity attributes.

A more complex but richer approach can be achieved by combining both concepts (bottom diagram of Figure 1). In fact, a Relying Party can use an IdP for hiding all user authentication details, and the IdP can interact with the PIdP for authenticating the user with his/her eID and get his/her PKC identity attributes.

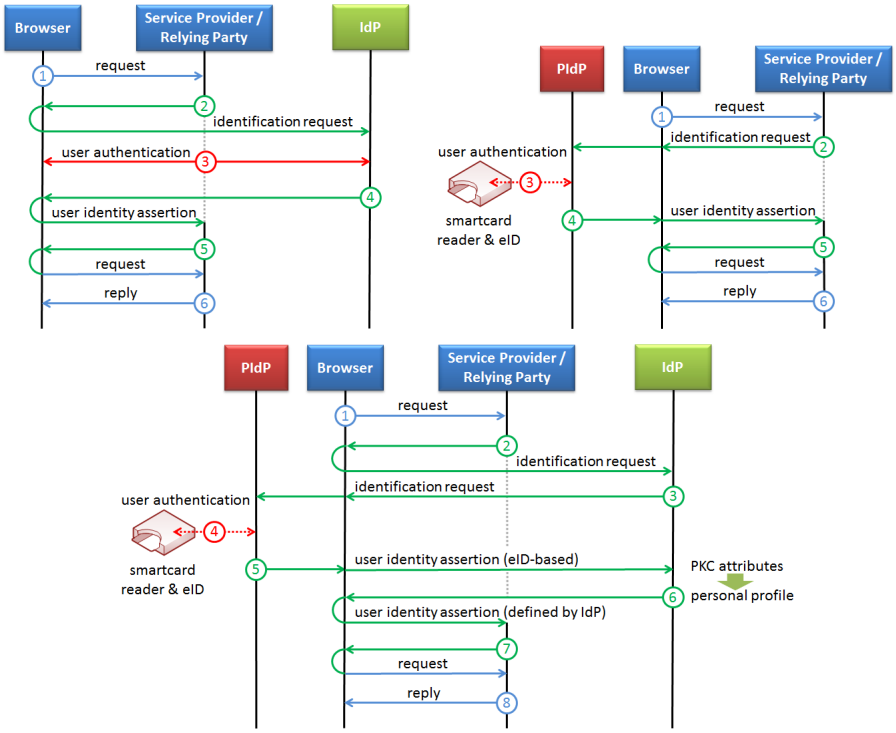
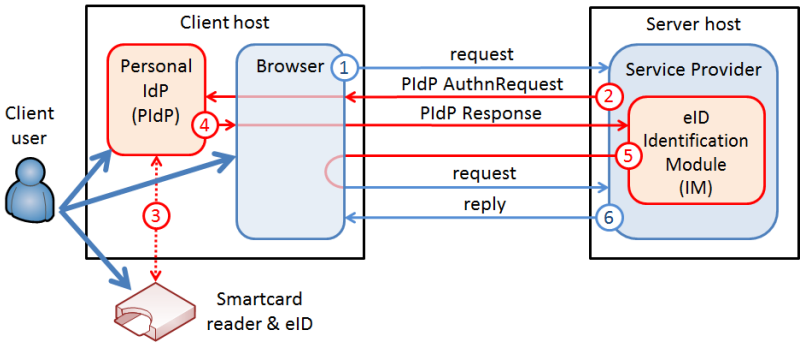


Fig. 1. Common IdP-based client identification architecture in Web interactions (top-left), alternative architecture using a PIdP and an eID (top-right) and a combined architecture using an IdP service and a PIdP (bottom).

4 PIdP-based identification and authentication

Our new identification architecture uses eID cards and a PIdP local to the person that wants to get identified (see Fig. 2). In terms of interface, the PIdP is a Web server. Along this text we will use the term *user* to refer the person that will be identified and authenticated by a service using his/her eID.

Whenever a service handling a client request requires the identification of the requesting user (1), it requires it to the client's PIdP (2), which must run on the client's host (thus, in the IP address 127.x.x.x). The identification request is made with a redirection of the client browser to its local PIdP, together with some authentication parameters. The PIdP interacts with the user in order to conduct his/her authentication with an eID (3), upon which it uses a parameter of message 2 to redirect the client browser to the service's identification module (IM) (4), which will validate the authentication proof and collect the user PKC, both provided by the PIdP. Upon success, the IM stores the user PKC in the proper session context, and redirects the client to its original target service (5), which will then provide the service to an already identified client (6).



2	service→PIdP : service name, PKC_s, r_1
3	PIdP →User : service name, PKC_s subject name User →PIdP : authentication PIN PIdP → eID : signature request, PIN, r_1, r_2, PKC_s eID →PIdP : $\mathrm{Sig} = \lceil r_1, r_2, \mathrm{PKC}_s \rfloor_{K_u^-}$ PIdP → eID : certificate request eID →PIdP : PKC_u
	PIdP : $K = \mathrm{digest}(r_1, r_2)$
4	PIdP → IM : $r_1, \{r_2\}_{K_S^+}, \mathrm{Sig}, \{\mathrm{PKC}_u\}_K$
	IM : validate r_1, recover r_2 and compute $K = \mathrm{digest}(r_1, r_2)$ IM : recover and validate PKC_u, validate Sig with PKC_u IM : $\mathrm{PKC}_u \rightarrow$ U's identity attributes

Fig. 2. Proposed user identification architecture (top), using a PIdP (Personal IdP) to handle the authentication and identification of a client user with his/her eID, and the protocol details (below), using the same interaction numbering. $\lceil x \rfloor_y$ stands for the "'the value x signed with private key y".

The PIdP is the only component that interacts with the eID and it does not need to be integrated with the client browser. It has its own graphical interface for interacting with the local user in order to customize the exploitation of the eID. The benefits of this approach are twofold: (1) the client cannot be fooled by a phony browser interface requiring credentials to use the user's eID; and (2) this approach works with all browsers (or other HTTP user agents) capable of supporting server replies containing HTTP redirections. This decision holds little impact over the security mechanism of the user eID, as it is basically an exploitation facilitation decision. The PIdP may very well use the native local interface of the eID middleware (for example, for getting an authorization PIN).

The PIdP is a local service of the user, therefore it doesn't need to be available to other machines in the Internet. Consequently, it can use a localhost IP address (e.g. 127.0.0.1). However, both this address and the TCP port used to receive HTTP requests have to be standard (in order to be known by the authentication requester). Alternatively, these elements can be conveyed by the browser as HTTP header fields.

We are assuming that eID owners explore this architecture in single-user, personal machines, and not in multi-user computers, because in those it would not be trivial to connect a personal eID reader for each user. This assumption rules out the need to distinguish among several PIdP services on the same host.

4.1 The identification and authentication protocol

The identification and authentication protocol is based on the signature provided by the eID, and the identity attributes of the user are extracted by the service's IM from the PKC that is used to validate the user's digital signature.

The protocol uses two random challenges, r_1 and r_2, from which a temporary key K is computed with a digest function. It also uses the users' private key (K_{U}^{-}) and the corresponding public key certificate (PKC_u), and the service's private key (K_{S}^{-}) and the corresponding public key certificate (PKC_s).

The challenges r_1 and r_2 are provided by each of the participants: the authenticating service and the user (his/her PIdP), respectively. These challenges, together with PKC_s, form a value that is signed by the eID; this signature will constitute the authentication proof presented by the eID owner. The validation of this proof must be performed with the public key corresponding to the private one that was used in the signature. This public key is provided embedded in PKC_u, also obtained from the eID. This certificate serves two purposes: (1) enable the IM to validate the signature with the proper public key, and to check the validity of the key pair, and (2) enable the IM to get, in a trustworthy way, identity attributes of the user (belonging to PKC_u).

The value of r_1 may optionally include a subcomponent with a service timestamp. This timestamp can help the IM to discard authentication requests happening after a threshold timeout defined solely by the service.

The purpose of the random value r_2 is to prevent a malicious service to collect specific r_1 values signed with users' private keys. The protection principle here is that the eID never signs a value that is solely provided by a third party; it must always include a random component generated locally by the PIdP.

4.2 Identification of services and protection of user's privacy

The user identity elements, contained in his/her PKC_u, should not be delivered to services in clear text for preserving the users' privacy from eavesdroppers. Furthermore, attackers running rogue services should not be able to impersonate legitimate services just to know the identity of clients.

To protect against these issues, all critical elements provided to the IM are protected with the service's public key, K_{S}^{+}, contained in PKC_s. Namely, the challenge r_2 is encrypted with K_{S}^{+}, thus can only be recovered by the owner of K_{S}^{-}, and the user's identity attributes, conveyed in his/her PKC_u, are encrypted with K, a temporary key derived from both r_1 and r_2. Therefore, a malicious service using a stolen PKC_s can not recover r_2 nor K, thus cannot observe PKC_u.

The services' certificates are not validated by the PIdP using certification chains or validity dates. Furthermore, they can be different from the certificates used by the services' underlying HTTPS servers. The PIdP should use a strategy similar to SSH [6] to handle relationships between services names and their certificates: if yet unknown, the user is asked whether or not it should be maintained; if already known, the PIdP checks if it has changed and, upon a change, asks the user if it authorizes the update of the existing information.

Note that the whole system is prepared to validate services' certificates according with X.509 rules at any time, but in our opinion it is excessive. As we will see, the main drawback of this approach is the possibility to provide a PKC_u to an attacker. This topic will be further addressed in the next section.

5 Security analysis

In this section we will evaluate the security of our authentication protocol regarding two kinds of attacks against authentication protocols: (i) identity stealing and (ii) identity surveillance.

A user may be fooled by a rogue service provider presenting a misleading identity (name and certificate). Two situations can occur: (i) the server presents a stolen certificate, for which it has no corresponding private key; or (i) the server presents its own certificate, for which it has a private key.

In the first case, the server would not be able to interpret the PIdP response, namely to get PKC_u, because it doesn't know K_{S}^-. For the exact same reason, a network eavesdropper would not also be able to interpret the user response, and therefore would not be able to get any information about the user identity.

In the second case, the server is able to interpret correctly the PIdP response, with his/her identity attributes. This allows the server to become aware of the user identity, but not to impersonate him/her, because the response cannot be reused (either as it is or after some manipulation). In fact, the values r_1 are different on each authentication run and the certificates PKC_s are different for each server. Therefore, a signature made by an eID once for an authentication process cannot be used in another authentication process (because either r_1 or PKC_s are different). We are assuming, of course, that given a signed piece of data it is impossible to find another one, different from the former, where the same signature fits. Therefore, the response provided by the PIdP when interacting with a particular authentication service cannot be stolen and reused with another server. This is true as well for a network eavesdropper, which has even less control over the PIdP response than a server being able to interpret the response.

Consequently, the protocol is secure against identity theft attempts based on stolen PIdP responses, either when the attacker does not know K_{S}^- (network eavesdroppers or rogue services using stolen certificates) or when the attacker knows K_{S}^- (rogue service using its own certificate). In the worst case, a rogue service may be able to learn the user identity after being able to convince the user to participate in a eID-based authentication. However, with password-based

systems this risk is incomparably higher, since in that case the rogue service can, in fact, steal the user's authentication credentials.

Finally, certificate validation would not help most users to protect themselves against false certificates presented by rogue services. In fact, the vast majority of users doesn't know anything about certificates and certification chains, thus they are not likely to react appropriately to error messages provided upon errors detected in the validation of certificates. Therefore, they can always be misled by rogue certificates when asked to proceed or not upon validation errors.

6 Implementation and experimentation

We have implemented a prototype of the authentication protocol using the PIdP and HTML mechanisms. The prototype was tested with the IdP-based identification infrastructure of University of Aveiro, which was adapted to perform the identification and authentication of the client using his/her PIdP and the Portuguese eID. The prototype was designed to work with all browsers, therefore we only used standard HTML mechanisms. Furthermore, we restrained ourselves from using facilities such as Java Script, because users and/or domain-wide policies may block them for security reasons.

The PIdP is a stand-alone, graphical application programmed in Java. It is based on the `HttpServer` class and uses Swing for managing the graphical interface. Upon a first authentication contact by a local client (browser), the PIdP launches a thread that conducts the interaction with the user and with his/her eID. In the meanwhile, the PIdP main thread keeps a dialog with the browser for showing a minimum of information regarding the current interaction with the eID. Once this interaction finishes, the PIdP redirects the browser to the validation service provided in the initial redirection URL.

The interaction with the eID uses native PKCS #11 C libraries. Regarding other crypto, r_1 and r_2 have 16 bytes each and K is computed with SHA-1 (first 16 bytes of the output). The symmetric encryption of PKC_u with K is with AES-128 (with ECB and PKCS5 padding). The encryption of r_2 with K_{S}^{+} is with RSA (ECB mode and OAEP with SHA-1 and MGF1 padding).

The authentication request to the IdP is an HTTP GET request to URL `http://127.0.0.1:[port]/authenticate` with 4 parameters: the 3 presented in the protocol description (see Fig. 2) plus the URL where the response should be returned; port will be discussed below. The authentication response is also an HTTP GET request, to the URL provided in the request, with the 4 parameters presented in the protocol description. All byte array parameters are transmitted encoded as hexadecimal strings.

The PIdP is addressed using a standard and uniform IP address for the client host (127.0.0.1) and a TCP port. The IP address does not pose any addressing issues, as it is interpreted by the client browser. On the contrary, dealing with the PIdP TCP port is more complex. According to standards [7], a port in a URI is expressed solely with digits; names are not allowed, therefore we could not rely on some existing port name resolver.

For our first experiments we chose and used a fixed port number (666) for the PIdP. As this is not an elegant solution, we are actually working on a name server using a well-known port number, which could appropriately redirect requests to local services (such as a PIdP). But for the sake of the experimental tests, the result is exactly the same.

7 Conclusions

In this paper we have presented a new approach for dealing with the Web authentication with eID tokens. This approach is based on a local application, called PIdP (Personal Identity Provider) that does not need to be trusted by the authenticator. On the other hand, the PIdP should be trusted by the eID owner, and in that sense our approach is more is more likely to fulfill this requisite than Java Applets or other form of dynamic code downloaded just-in-time by browsers. Finally, users are free to choose, or even develop, their own PIdP. This means that they have the power to define which application interacts with their eID, which is an advantage for managing personalization features (e.g. PIN recording for particular services) and for preventing abusive interactions with eIDs (e.g. multiple signature requests instead of a single one).

The PIdP-based authentication architecture is completely independent from browsers and can be used directly by service providers or by identity providers. Our prototype, based on a Java PIdP and on Java Servlets, was tested in both scenarios. Currently it works only with the Portuguese eID, but it can be easily extended to deal with eIDs of other countries or even companies. Furthermore, we can use other alternative physical locations of the user credentials (files, USB dongles, etc.) other than an eID, because it is up to the PIdP to look for them.

References

1. Hughes, J., Cantor, S., Hodges, J., Hirsch, F., Mishra, P., Philpott, R., Maler, E.: Profiles for the OASIS Security Assertion Markup Language (SAML) 2.0. OASIS. (March 2005) `http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf`.
2. Rescorla, E.: HTTP Over TLS. RFC 2818 (Informational) (May 2000)
3. Bour, I.: Electronic Identities in Europe: overview of E-ID solutions connecting citizens to public authorities. UL Transaction Security Whitepaper (April 2013)
4. Cooper, D., Santesson, S., Farrell, S., Boeyen, S., Housley, R., Polk, W.: Internet X.509 Public Key Infrastructure Certificate and Certificate Revocation List (CRL) Profile. RFC 5280 (May 2008)
5. Verhaeghe, P., Lapon, J., Decker, B.D., Naessens, V., Verslype, K.: Security and Privacy Improvements for the Belgian eID Technology. In: 24th IFIP TC 11 Int. Information Security Conf. (SEC 2009), Paphos, Cyprus (2009) 237–247
6. Ylonen, T., Lonvick, C.: The Secure Shell (SSH) Protocol Architecture. RFC 4251 (January 2006)
7. Berners-Lee, T., Fielding, R., Masinter, L.: Uniform Resource Identifier (URI): Generic Syntax. RFC 3986 (January 2005)