Anonymous Credentials in Web Applications
– A Child's Play with the PRIME Core

Benjamin Kellermann and Immanuel Scholz

Technische Universität Dresden
Faculty of Computer Science
D-01062 Dresden, Germany
{Benjamin.Kellermann|Immanuel.Scholz}@tu-dresden.de

Abstract. Web applications dealing with personal data in a privacy-friendly way have the need for anonymous credential systems. While there are already protocols describing anonymous credential systems and libraries, implementing the protocols, application using the libraries are rare. Without applications supporting anonymous credentials, companies will not start building a credential infrastructure and vice versa. This paper presents an easy way to issue and use anonymous credentials for web applications. By reducing the initial cost for both parties, the barrier of "starting first" can be lowered.

Key words: anonymous credentials, security programming

1 Introduction

Imagine a web application dealing with some personal data. It lets the user register and enter his age and nationality as well as a username and password for access control. The service operator does not want to worry about checking the accuracy of the personal data, so he uses a third party to certify these attributes. This kind of application has some disadvantages, e. g., in a naive implementation, the third party learns about the users intention to use the service. Additionally, the access control credentials (username and password) could be given to other people and finally the user is traceable through different sessions. All these problems can be avoided with anonymous credentials, introduced by Chaum [1].

Anonymous credentials, presented by Camenisch and Lysyanskaya [2], provide several features not present in "classic" credential systems. They are unlinkable, i. e., two subsequent presentations of the same credentials can not be linked with each other. Partial information on the attributes can be released, which means if a credential contains multiple entries (e. g., "age is 20", "gender is male" and "first name is John"), only some can be revealed, hiding the remaining entries. Relational proofs can be used for numerical entries (e. g., if a credential states "age is 20", the relation "age is greater than 18" can be shown without revealing the actual value.) Finally, they offer the so called "all-or-nothing sharing", which means sharing one credential leads to sharing all of the owners credentials, making it unattractive for users to disclose their credential information to others.

©IFIP, (2010). This is the author's version of the work. It is posted here by permission of IFIP for your personal use. Not for redistribution. The definitive version was published in M. Bezzi et al. (Eds.): PrimeLife Summer School 2009, revised and selected papers, AICT 320, pp 237–245, 2010, (Springer).

Many developers believe that implementing access control via anonymous credentials and building an infrastructure for credential issuers is very complicated. From a first point of view, it looks like a "hen-and-egg" problem. Companies, will not start to issue anonymous credentials without applications using them. Application developers will not start implementing access control via anonymous credentials, when there is no infrastructure issuing them.

This paper presents an easy way to enroll and verify anonymous credentials with the PRIME core [3], which uses the Idemix library [4]. This library already implements many features of the Camenisch-Lysyanskaya credentials system. A larger tutorial has been created during the development which shows additional features not covered in this paper [5].

The document is structured as follows. Section 2 explains the communication flow, which is done by the basic scenario and discusses, what has to be done to set up everything on the users side. The few web application modifications, which have to be done to use anonymous credentials with the PRIME core are shown in Section 3 and the steps to issue credentials are given in Section 4.

2 Setup

2.1 Architectural Overview

The setup and communication flow is shown in Figure 1. One can see that in addition to the users web browser and the web application, two other instances are needed. These are needed to exchange data through cryptographic protocols. We call these instances "PRIME core". They can be compared to PGP[1], in a scenario, where an E-mail application wants to send an encrypted E-mail. Two small programs, which intercepts the normal communication flow to do the cryptography, are needed there as well. Note that the communication is always initiated by the client, to ensure connectivity from behind a firewall or NAT.

Users are solely concerned with the client PRIME core installation. Administrators of issuer services and developers also have to cope with running and configuring a server. We will explain in the following, how a PRIME core is launched at client side. The administrators and developers perspective is considered at the beginning of the Sections 3 and 4.

2.2 Running PRIME on Client Side

Using the PRIME core at the users side is pretty much the same as running any other program. After unpacking the archive `prime.zip`, the `prime.jar` can be run without any parameter. This provides a tray icon with several menu functions to execute the different user functions.

There are two possibilities to deflect his web traffic through the PRIME core. The PRIME core may act as web proxy, which can be configured within the

[1] Pretty Good Privacy. An application for encrypting data, especially E-mails.

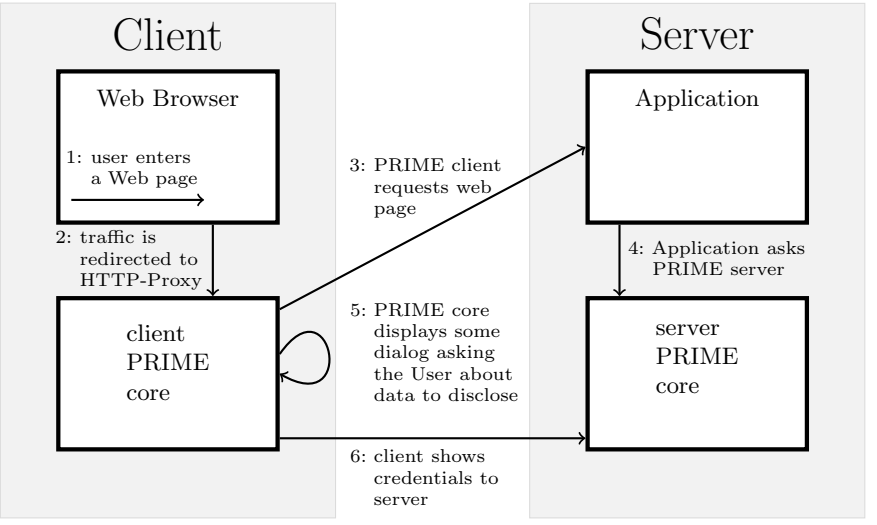

Fig. 1. Overview of the communication flow with PRIME. The arrows indicate the initiation of communication links.

config interface. The other possibility is to install a Firefox extension, which watches the traffic and calls the PRIME core if needed.[2]

3 How to Use PRIME in an Application

3.1 Launching PRIME as a Developer

Each PRIME core offers its functionality through a set of web services. Common web services are launched by default, which are sufficient, if the PRIME core should act as client. At server side, additional web services have to be launched as well as the graphical user interface has to be disabled. Because of disabling the user interface, passwords for some components (e. g., the Java secure key store) cannot be entered interactively. Therefore, the required passwords have to be given on the command line or in a configuration file.

Some of the PRIME core's web services are only launched when a password has been specified for them to access. These passwords are there to authenticate external programs accessing the PRIME core. They should not be confused with credentials or data that may have to be provided to access personal information within the PRIME data storage. If an application wants to access personal information, it may have to provide additional authorization information.

The web services are grouped in categories (like "common", "system", or "simplepolicy"). The mechanism of launching a category which is not launched by default is specifying a password for it. This ensures that

[2] This requires using Firefox of course, but it has other advantages, which might be desirable.

1. no default password leaves unprotected services unintentionally open and
2. wrong or unknown passwords are excluded as a point of failure when setting up the system.

A typical call, sufficient to launch the PRIME core for the needs of access control would look like:

```
java -jar prime.jar --gui=false --keystore.password=XYZ \
  --webservice.simplepolicy.password=YZX
```

The PRIME core offers a built-in developer help system for its web services. Directing a web browser to `https://localhost:9906`,[3] one can see a web page, describing which services are launched. Behind every function, a question mark is displayed, with which one can access an on-line help in form of a short description and a form to easily access the web service. This is shown in Figure 2.

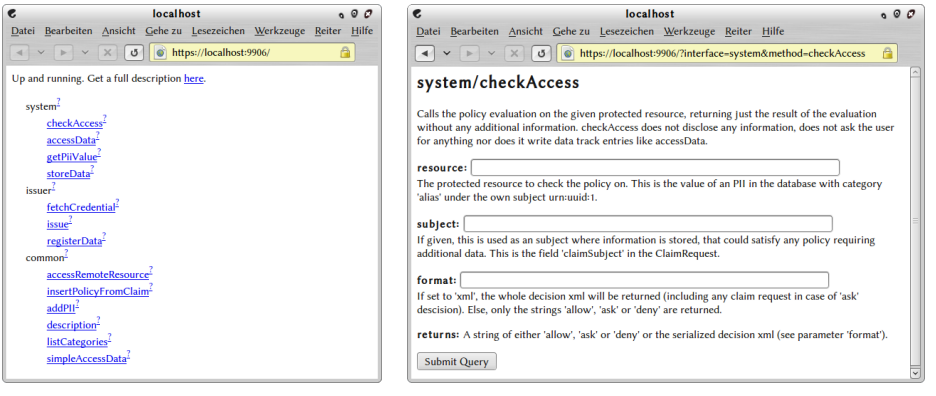

Fig. 2. Two help pages offered by the PRIME core. The overview page (left) and the description of `checkAccess` (right).

For debugging and developing, a set of very powerful `debug` web services is available, including a direct SQL access to the database and example implementations for typical server administrative services, like configuring policies or approving credentials.

3.2 PRIME enabled "Hello World!"

Assume a very simple "Hello World" web application consisting of one line of code (Figure 3). This application does nothing more than printing out the string "Hello World!", which should be the placeholder for a point, where access control is checked in a more complex application.

[3] For the rest of the document it is assumed, that the server runs on `localhost` and uses the default port `9906`.

```
1 <?php echo "Hello World!"; ?>
```

Fig. 3. Hello world web application in PHP.

Now, we want to implement access control to our "Hello World!" application by means of an anonymous credential. For this, we have to

1. create a policy in the server's database, and
2. modify the source code so that it asks the PRIME core to evaluate the policy and grant or deny access.

Inserting a Policy A policy requires a so-called "protected resource" which defines what data item the policy is about. The protected resource can be any URI[4] chosen by the developer – for example the URL of the web site to protect. This is also called "object", or just "resource" in some policy languages. In our example we use the self-chosen URI "urn:hello" as identifier for the protected resource.

PRIME supports very sophisticated policies. Different actions like read and write are supported. Policy rules can simply depend on the disclosure of any data or the data can be required to have a specific value. For numerical data like the age, relations can be specified (e. g., greater than 18 years old). Developer-specific relations are possible as well. It can be required that disclosed data is certified by anonymous or non-anonymous credentials. Data handling policies can be attached to data categories to specify meta information like the intended purpose or time period which the disclosed data is used for. These policies are specified in an XML policy language using the `policy/insert` web service. For the most common case "require existence of one data category", the much simpler web service `simplepolicy/insert` can be used.[5] After choosing the data category, which a user has to show to get access, the policy is ready for use.

Modifying the Source Code We already illustrated the communication flow of the application in Figure 1. Figure 4 tries to show this in more detail. When a user tries to access the web page (arrow 1 and 2 of Figure 1, or the first two "GET URL" arrows of Figure 4), the web application will ask the server-side PRIME core, if the user is allowed or not. The web service `system/checkAccess` can be used for this policy evaluation.[6] It returns one of three possible values: "allow",

[4] Uniform Resource Identifier. A character string to identify a resource.

[5] The on-line help (question mark next to the function) provides a simple web interface for sending the request (cp., Section 3.1). However, if you like to use a more convenient reference implementation, browsing to `https://localhost:9906/debug/managePolicies` tries to show how a simple policy manager may look like.

[6] The on-line help can be used again to quickly check the policy while developing (cp., Section 3.1).

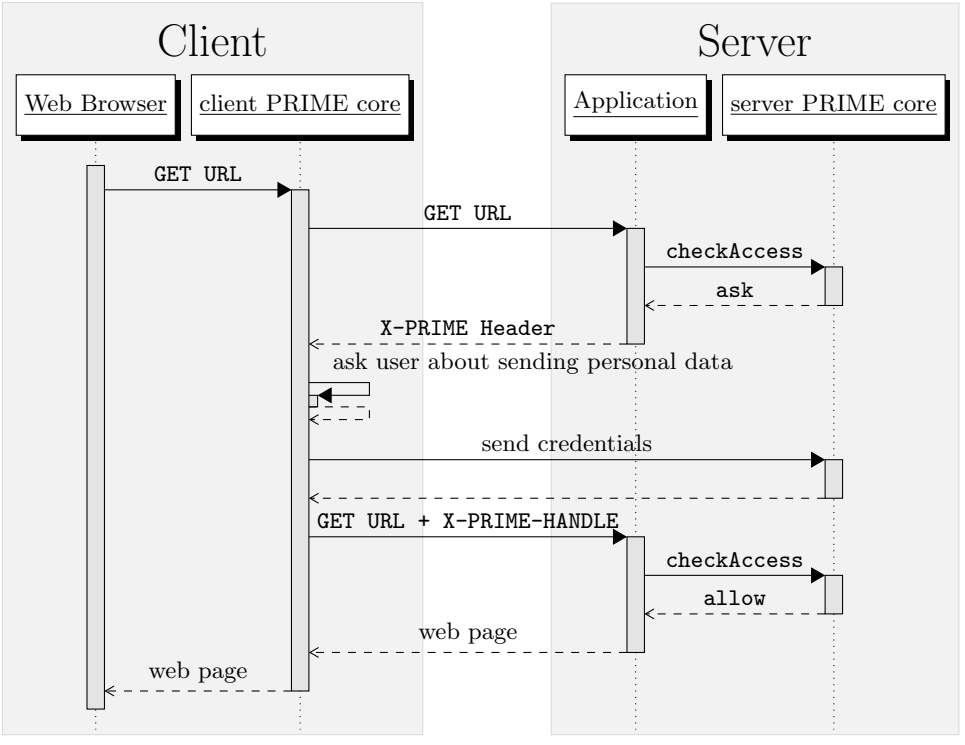

Fig. 4. Sequence diagram of the communication flow with PRIME.

"deny", and "ask". The state "ask" denotes that not enough information has been provided and the authorization should be started. Most developers using access control frameworks are only familiar with the two states: "allow" and "deny". Usually, the time when authentication checks are done has to be known to the developed application and some kind of logged-in state is maintained. It is possible but not encouraged with the PRIME framework to track and maintain a logged-in state (which usually comes with linkability between individual user transactions). However the best practice approach here is, to let the policy evaluation decide directly for each individual transaction. If this is technically feasible, it enables that a user's activities become unlinkable, say the browsing for books (remain anonymous) and the actual ordering (disclosing contact information).

For interpreting the three return values in PHP, a `switch` environment can be used. In case of "allow" or "deny", the server's output is `"Hello World!"` or any error message, respectively. In case of "ask", the policy evaluation determined that it cannot decide, yet, whether the user is allowed to access the protected resource (this is also the case in Figure 4). It needs more information, e. g., a proof of possession of an anonymous credential. This proof can be triggered by the server by inserting two HTTP-headers in the response to the client: "X-PRIME" and "X-PRIME-Protected-Resource". The first header defines the

address, the client should contact to show his credential, and the second one states the protected resource under which the policy is stored. The client-side PRIME core will interpret these headers, trigger the credential proving process with the client-side PRIME core ("ask user" and "send credential" arrows of Figure 4 or arrows 5 and 6 of Figure 1), and repeat the HTTP request to the web server including another HTTP header "X-PRIME-HANDLE", which contains a session id. The web application has to pass the session id to the server-side PRIME core, which evaluates the policy again and now hopefully returns "allow".

Figure 5 shows the complete PRIME-enabled "Hello World!" application. Lines 2–4 queries the web service `system/checkAccess` of the server-side PRIME core. Lines 5 and 6 reply the easy cases, where access is allowed or denied. Lines 7–10 return HTTP-headers which causes the client to show the required credentials.

```
1  <?php
2  switch (fread(fopen("https://localhost:9906/"
3          ."system/checkAccess?resource=urn:hello"
4          ."&subject=$_SERVER[HTTP_X_PRIME_HANDLE]","r"),10)){
5  case "allow": echo "Hello World!";  break;
6  case "deny":  echo "Access Denied"; break;
7  case "ask":
8          header("X-PRIME: https://example.org:9906");
9          header("X-PRIME-Protected-Resource: urn:hello");
10         break;
11 }?>
```

Fig. 5. Example source code of the PRIME enabled "Hello World!" application.

With these additional 10 lines of code, the developer has integrated whole access control and credential verification features of the PRIME core.

4 Credential Issuing

4.1 Running PRIME as Issuer

If one wants to issue credentials, the `restricted` web services have to be launched in addition to the ones one has to launch for access control. An appropriate command line to launch the server could look like:

```
java -jar prime.jar --gui=false --keystore.password=XYZ \
  --webservice.simplepolicy.password=YZX \
  --webservice.restricted.password=ZYX
```

Also, some configuration has to be done to specify for which cryptographic key and which data categories the service will issue credentials. A reference implementation

has been made within the `debug` web services. Browsing to `https://localhost:9906/debug/configIssuer` will provide an easy click-through interface, which does the necessary configuration for issuing credentials.

4.2 Issuing Credentials

Issuing of credentials is one of the functionalities, built into the PRIME core. There is no need for developers to do additional programming to offer credential issuing for end-users. Steps to a successful issue a credential are

1. submitting data to be certified to the issuing service,
2. convincing the service provider about the correctness of the data[7], and
3. fetching a credential for this data.

Submitting Data To fetch any credential, the option "Fetch Credential" in the send personal data dialog can be used.[8] Alternatively, the fetching can be initiated by the menu option "Register Data" in the tray icon.[9]

After entering and sending the data, a 4-digit number is displayed. This number is needed in the next step.

Convincing the Issuer During the registration process, a shared secret[10] is stored at the client and the server. A 4-digit hash of this secret is displayed to the user as described previously. By revealing this hash value, the user proves to the credential issuer that the data at the server was indeed provided by his client computer. The 4-digit hash is used here for usability reasons, a longer value or even the whole secret can be used in other scenarios.

When the credential issuer is convinced, that

- the person is authorized to get a credential for this data (e. g., by verifying the id-card) and
- that the credential in the server database is the credential from the persons' computer (by checking if the data stored under the shared secret is correct),[11]

he approves the credential by calling to the web service `restricted/setProven` and informs the user, that his client can fetch it.

A minimalistic sample implementation has been realized within the `debug` web services. Browsing to `https://localhost:9906/debug/managUnprovenPii` delivers an easy interface which displays all unapproved hashes and personal data together with a link to approve them.

[7] In practice it may often be the case that the issuing party already has the data it wants to issue. However, the user still has to convince the service provider, that the data was submitted from his personal device.

[8] The send personal data dialog is the dialog that pops up when browsing on a protected web site with a PRIME-enabled web browser.

[9] In this case, the issuing service URL has to be specified manually.

[10] Here, the shared secret is a 122 bit long random number.

[11] Imagine an attacker, submitting the same information to trick the clerk verifying the id-card into approving his credential request instead.

Fetching the Credential After approving the request, the user can obtain the credential via the "Fetch Credential" menu option in the clients tray icon.

5 Conclusion

We showed, that anonymous credentials are easy to handle with the PRIME core. Most of the features of modern anonymous credential systems were provided without any additional effort. Obviously, the authentication with the PRIME core could be implemented in addition or as an alternative to an existing one, which improves acceptance and ability to integrate PRIME into existing projects.

Our simple example showed a way to present an unlinkable, partially-provable credential which also supports relational proofs. If access control is bundled at one point of the application, about 10 lines of code are necessary to replace the normal authentication.

Acknowledgments. The authors want to thank Mike Bergmann, Sebastian Clauss, Martin Meinhold, and many others for the development on the PRIME core. Helpful comments from Rainer Böhme, Sebastian Clauss and Stefan Köpsell have been incorporated. In addition, we want to thank Jan Camenisch and the other anonymous reviewers.

The research leading to these results has received funding from the European Community's Seventh Framework Programme (FP7/2007–2013) under grant agreement №216483. The information in this document is provided "as is", and no guarantee or warranty is given that the information is fit for any particular purpose. The above referenced consortium members shall have no liability for damages of any kind including without limitation direct, special, indirect, or consequential damages that may result from the use of these materials subject to any liability which is mandatory due to applicable law. Copyright 2009 by TU Dresden.

References

1. Chaum, D.: Security without identification: Transaction systems to make big brother obsolete. Commun. ACM **28**(10) (1985) 1030–1044
2. Camenisch, J., Lysyanskaya, A.: An efficient system for non-transferable anonymous credentials with optional anonymity revocation. In Pfitzmann, B., ed.: EUROCRYPT. Volume 2045 of Lecture Notes in Computer Science., Springer (2001) 93–118
3. Casassa-Mont, M., Crosta, S., Kriegelstein, T., Sommer, D.: Architecture v2. Deliverable D14.2.c, PRIME (March 2007) `https://www.prime-project.eu/prime_products/reports/arch/pub_del_D14.2.c_ec_WP14.2_v1_Final.pdf`.
4. IBM Research: Identity mixer. `http://prime.inf.tu-dresden.de/idemix/` (November 2009)
5. Kellermann, B., Scholz, I., Wahrig, H.: The PRIME developers tutorial. `http://turrican.inf.tu-dresden.de/doc/` (August 2009)