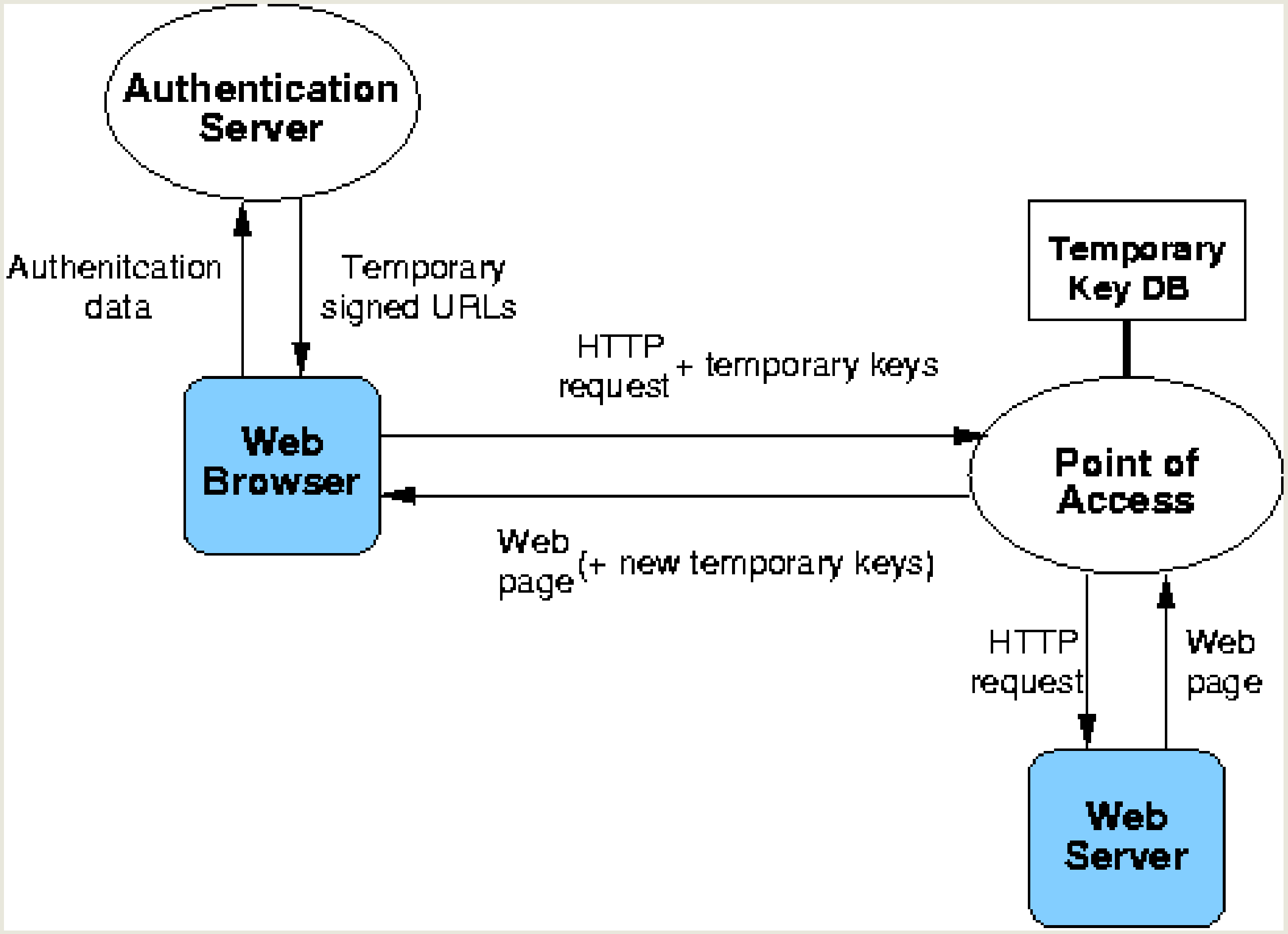
PAPI and LDAP

Using directories for local authentication and authorization

Diego R. López

diego.lopez@rediris.es

The PAPI architecture
An overview



PAPI phases
And when LDAP is used

- **Authentication (at the AS)**
  - User is identified
  - Assertions to be sent to the different PoAs are generated
- **Authorization (at the PoA)**
  - Assertions coming from the AS are validated
    - Temporary tokens (cookies) are generated and stored
  - Temporary tokens are received
    - Fresh tokens are generated and stored if needed
- **LDAP is currently employed for user identification and assertion generation**
  - Ongoing work for refining assertion generation and their validation at the PoAs

User identification

- **Users are identified by means of a bind operation**
  - The DN is derived from the "username" the user provides in the authentication form
  - The same DN is used for building the assertions
  - Only simple authentication is supported
    - Data are transferred to the AS using SSL
- **Next version will include identification procedures based on X.509 certificates**
  - The DN in the certificate will be the one used for building the assertions

Assertion generation
The papi* classes

- **The assertion procedures build them using the DN derived from user input**
  - Using the attributes of the `papiUser` class
    - The groups the user belongs to
      - A list of identifiers in the `papiGroupId` attribute
    - The sites the user has explicit access to
      - A list of identifiers in the `papiSiteId` attribute
    - `papiGroup` objects also contain a list of sites in their `papiSiteId` attributes
  - The final outcome of this process is a list of `papiSite` objects
    - Obtained as the union of explicit and implicit site references

Assertion generation
The papiSite class

- **Contains the definition of a PAPI PoA**
  - The URL of the PoA
  - The location for assertion validation at the PoA
  - The time to live to be requested for the tokens
  - The service identfier used by the PoA
  - A description of the service to be accessed
- **The assertion procedures build the URLs for requesting access through each of these PoAs using:**
  - The data read from the `papiSite` class
  - The data derived from the user LDAP entry to identify her/him at the PoA
    - As returned by the user identification function

Assertion generation
Controlling IDs sent to the PoA(s)

- **The current implementation sends the same ID to any PoA it contacts**
  - Too coarse
  - Little user control on privacy preservation
- **A new attribute in the `papi*` classes will allow for defining the contents of the ID**
  - Define a specific format for a `papiUser`
  - Define a common format for a `papiGroup`
  - Define a default format for a `papiSite`
  - Include free text and references to attributes in the `papi*` class
- **Currently defining the (XML-based) format and precedence rules**

PoA configuration
Reducing complexity

- **Experience shows that the number of PAPI PoAs at any installation tends to be high**
  - This is why GPoAs are defined in PAPI 1.1
- **Configuring a PoA requires a set of values to be included into the Apache configuration**
  - Many configuration values are common among PoAs
    - In the same or another server
  - Updating them requires:
    - Priviliged access to all servers
    - Repetitive (and thus error-prone) procedures
- **An obvious solution for this is to have LDAP-based PoA configurations**

PoA configuration
What to put in the directory

- **Almost everything in a PAPI PoA configuration can be stored (and shared) using LDAP**
  - AS pubkeys (=> certificates)
  - Time-outs
  - Locations in URLs
  - GPoA definitions
  - Filters
  - Proxy-mode configuration
- **The only possible exceptions are file locations**
  - Including private keys and databases
  - Although they could be used as "standard" values

Assertion evaluation

- **Assertions from the AS are statically evaluated at the PoA**
  - Based on filter specifications
  - Changes in user rights are not propagated until re-authentication occurs
- **A PoA could dynamically evaluate the assertion using the ID inside it**
  - As a handle to an attribute server that enforces privacy preservation policies (a la Shibboleth)
  - As an anonymized reference to a directory entry
    - Privacy policies can be enforced by directory ACLs
  - A remote call to the Policy Engine API