A Detailed Description of the PAPI Protocol

The PAPI Development Team <papones@rediris.es>

Contents

1 An overview of PAPI

PAPI is a system for providing access control to restricted information resources across the Internet. It intends to keep authentication as an issue local to the organization the user belongs to, while leaving the information providers full control over the resources they offer.

The authentication mechanisms are designed to be as flexible as possible, allowing each organization to use its own authentication schema, keeping user privacy, while offering target sites enough data to perform authorization and collect usage statistics. Moreover, access control mechanisms are transparent to the user and fully compatible with the most commonly employed Web browser and any operating system.

The system consists of two independent elements: the *Authentication Server* (AS) and the *Point of Access* (PoA). This structure makes the final system much more flexible and able to be integrated to different environments. There

is no need of a one-to-one mapping between ASes and PoAs: a given PoA may manage to deal with requests from any number of ASes and direct them to any number of web servers.

The purpose of the AS is to provide users with a single authentication point and enable them to retrieve (in a completely transparent manner) the temporary keys that will let them access the services they are authorized to. A PAPI2 AS may also include an Attribute Authority (AA) server associated to it, able to provide requesting PoAs with those relevant attributes related to the user that is trying to access a given resource.

The PoA manages authorization and actual access control to a set of web locations for a given organization. The owner of the target site have the responsibility of managing this point of access. A PAPI PoA can be adapted to any web server, whatever its implementation is. Moreover, a given web server can have more than one PoA, and a PoA can control more than one web server. PoAs can be hierarchically combined into groups controlled by a *Group-wide PoA* (a GPoA), a trusted source within the group for assessing user data without requiring them directly from the ASes.

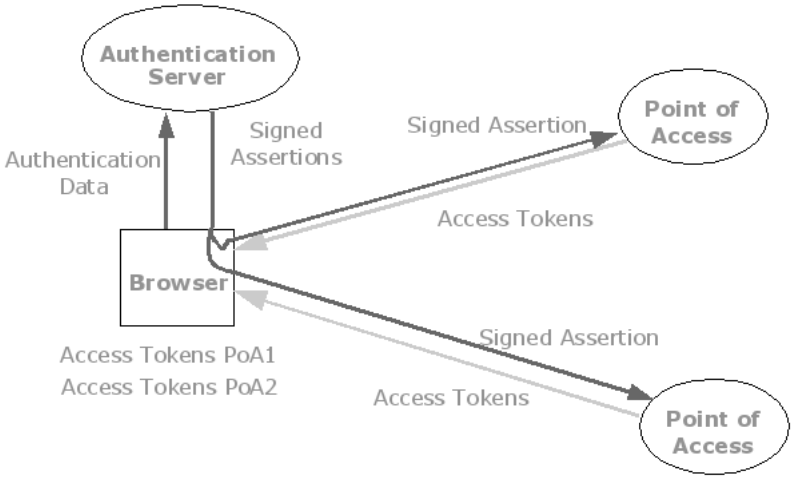

Figure 1: The PAPI base protocol

1.1 The PAPI protocol in brief

With PAPI, authentication is a local matter, and authorization too. Authentication occurs at the user's organization, possibly accessing data that must not be disclosed in any case. Once authenticated, the user is able to go the entry points of the PoAs. It is important to remark that the AS is not sending any user-provided data to any PoA. It prepares an assertion (as required by the PoA) about the user and signs it using its private key. The only constraint that a PoA imposes on the assertion about an user sent by an AS is that the

identifier must be unique at least during the lifetime of the tokens the PoA is going to provide. Of course, information should be also enough to pass through the rules the PoA enforces, but the AS is never required to disclose any private information.

The PoA receives this chunk of information, signed by the AS, and decides whether to grant access to the user or not. It is important to note that when we refer to a "PoA trusting an AS", we are not talking about a PoA permitting any access request coming from that AS, but about the PoA trusting the assertions the AS makes. That means that, if a PoA trusts an AS, the (digitally signed) assertion of "This is user X of group Y" made by the AS is going to be trusted by the PoA. And the PoA decides, according to the assertion and its policy, to grant access or not. Authorization is, again, a local matter to the organization operating the PoA.

In the following sections we will describe the interactions among the different elements of the system, and the data exchanged by them. The PAPI protocol has three different phases:

1. Authentication, that takes place at the AS, in which users provide proofs of their identity and make the AS get ready to produce the appropriate assertions for them.
2. Authorization, that occurs at the PoA the first time it is contacted within a session. In this phase, the PoA seeks for assertions about the requesting user and evaluates them, deciding whether to grant access or not.
3. Access control, carried by the PoA when processing requests once the initial request has been authorized.

2 PAPI application scenarios

The decoupling of PAPI components and the different interactions that the protocol offer permits the existence of different scenarios, depending on the flow of user actions and the exchange of data among the components of the system. Let's consider the following components, that define the simplest configuration containing all the possible interactions:

- An Authentication Server, AS0, located as `http://as0.source.domain/AuthServer`, able to validate user credentials and containing a definition of the PoAs able to accept its assertions.
- A Point of Access, PoA0, providing access to the Web location `http://poa0.target0.domain/`
- A Group-wide Point of Access, GPoA0, located at `http://gpoa0.target1.domain/` and not providing access control to any Web resource.
- A couple of Point of Access, PoA1 and PoA2, providing access to the Web locations `http://poa1.target1.domain/` and `http://poa2.target2.domain/`, and subordinated to GPoA0. Note that, while PoA1 is in the same domain as GPoA0, that is not the case for PoA2.

Since GPoA0 acts as trust aggregator for PoA1 and PoA2, there is no need for AS0 to know about them. As far as the Authentication Server is concerned, its trust links are maintained with GPoA0, and as many subordinated PoAs as required may be defined under GPoA0 without affecting the other components.

Four different scenarios can be considered:

1. The user first authenticates at AS0, which answers with a page containing links to the available PoAs known to it.
2. After authentication at AS0, the user tries to access a PoA under GPoA0.
3. Without prior authentication, the user tries to access either PoA0 or a PoA under GPoA0.
4. Once it has accessed the resource intended in 3, the user goes to some other resource.

2.1 The access control phase

These four scenarios differ in their authentication and authorization phases. Access control is always performed by means of a pair of cookies, named `Hcook` and `Lcook`, that are passed to the user's browser when authorization succeeds. These cookies are free-format strings containing the following data:

`Hcook` `isTemporary:userData:expiryTime:randomBlock:location:serviceID`

`Lcook` `timeStamp:location:serviceID:userData`

Where:

`isTemporary` Is a boolean value that defines if the Hcook is temporary or persistent. Temporary Hcooks are used in PoAs that depend of a GPoA.

`userData` Contains the information about the user kept by the PoA after authorization, derived from the assertion received by the PoA. May be passed (for those URIs matching the regular expression defined by the PAPI configuration directive `Hcook_Handler`) to other elements processing the request by means of either an Apache note called `PAPIHcook`, or a HTTP request parameter called `X-PAPIHcook`.

`expiryTime` Sets the moment in which this Hcook will no longer be valid, expressed in seconds since the epoch (1 January 1970).

`randomBlock` Is used to check the value of Hcook with the cookie register, in order to avoid unauthorized access by cookie copying (see below).

`location` Is the Apache location the PoA is defined for.

`serviceID` Is an identifier to select the PoA among others defined for the Apache server it is running on.

`timeStamp` Is the moment in which this value of Lcook was generated

These values are encoded using the AES symmetric-key algorithm. The keys employed are stored as strings of up to 32 hexadecimal digits inside the files defined by the PAPI configuration directives `HKEY_File` and `LKEY_File`, respectively. Hcook is sent to the browser so as to make it permanent (storing its value in disk) by means of a `expires` parameter in the `Set-Cookie` HTTP header , while Lcook does not use this parameter and is therefore kept by the browser just in memory. Thus Lcook is intended to be used as a session cookie, while Hcook is used to keep the authorization state between sessions.

Whenever a request arrives for a resource under a PAPI-protected location, the PoA checks for these cookies. If they are not found, an unauthorized access error is raised. If the cookies are found, the PoA first decodes and analyzes Lcook to check:

1. Whether it has not expired, according to the PAPI configuration directive `Lcook_Timeout`.
2. If there are no immediate filters applicable to the user data, as defined by the PAPI configuration directive `Cookie_Reject`.
3. If the rest of elements (`location` and `serviceID`) are correct.

If everything is correct, access is granted. If there is a failure in steps 2 or 3, a fatal access error is raised. In the case that Lcook has expired, the PoA decodes and analyzes Hcook, verifying that:

1. It has not expired, using the value of its `expiryTime` element.
2. There are no immediate filters applicable to the user data, as defined by the PAPI configuration directive `Cookie_Reject`.
3. The rest of elements (`location` and `serviceID`) are correct.
4. The `randomBlock` corresponds to the one stored in the cookie registry for this location, user data and service identifier.

If the first test fails, an unauthorized access error is raised. If any of the other tests fails, a fatal access error is raised. If the tests pass, access is granted, and a fresh set of cookies are sent back along with the response to the user's browser. The cookie registry is updated accordingly, using the new random block generated for the new Hcook.

When the PoA detects a fatal access error, the Apache server is directed to simply deny the access using a 403 HTTP response code. If an unauthorized access error is signaled, the behavior of the PoA depends on its configuration. If a GPoA is defined by means of the `GPoA_URL` configuration directive, the request is sent upwards within the GPoA hierarchy in order to ask the parent GPoA for authorization. If no GPoA is defined, the Apache server will deny access using a 403 HTTP response code.

2.2 Scenario 1: Authentication enabling direct access

When the user first accesses AS0 at `http://as0.source.domain/AuthServer`, it receives a login page to provide their authentication data (username/password pair, a certificate,...). These data are sent to AS0 by the user (activating the submit of a form in the page, selecting a certificate in the browser,...), which validates them according to its configuration. If authentication data is not accepted, a reject page is sent back to the user.

Once authentication data has been accepted, AS0 looks for the sites the user has rights to access and that have to be contacted upon successful authentication. Assume it founds that they are PoA0 and GPoA0. AS0 prepares an accept page to be sent back to the user. Along with this page, AS0 sends to the browser an authentication cookie, named `PAPIAuthNHcook` and encrypted using the AS key, that will be used for automatic re-authentication if necessary (see section 2.5). Inside this HTML page, AS0 inserts references (through URLs in HTML tags such as `<img>`, `<script>`, `<style>`, etc.) to PoA0 and GPoA0. Inside these references, the assertions about the user to be sent to PoA0 and GPoA0 are encoded and signed with the AS0 private key. These URLs looks like this:

`http://poa0.target0.domain/AuthLocation?AS=AS0&ACTION=LOGIN&DATA=absc5...`

`http://gpoa0.target1.domain/AuthLocation?AS=AS0&ACTION=LOGIN&DATA=wqrt...`

Note that the `DATA` parameter is different, since the assertion sent to each PoA may be different, according to the AS configuration.

The general format of the `DATA` parameter is:

`userAssertion:expiryTime:currentTime:location:serviceID`

When the user's browser receives this page, it tries to fulfill the data to be presented to the user by contacting PoA0 and GPoA0 according to the URLs inside the above mentioned HTML tags. When PoA0 and GPoA0 receive the request, they evaluate the request to:

1. Check whether an AS named AS0 is recognized by them, that is, it is defined by a `PAPI_AS` directive and there is a file named `AS0_pubkey.pem` in the directory defined by `Pubkeys_Path`.
2. Verify if the data sent by AS0 is correctly signed, by decrypting it with the key found in the previous step and checking its format.
3. Verify that the assertion about the user has not expired (`currentTime` is inside the limits defined by the parameter `URL_Timeout`), and that `location` and `serviceID` are correct.
4. Check if the user assertion matches one of the filters defined by means of `PAPI_Filter` and whether if translates into an `accept` or a `reject` result.
5. If an external authorization engine has been defined by means of `SPOCP_Server`, prepare a query to the authorization server using the assertion about the user and send it to the server.

In the case one of them finds the request unacceptable, it sends back to the user's browser a particular object (image, JavaScript snippet, CSS stylesheet,...) according to the value of the `RURL` parameter of the request (or a default object configured in the PoA by means of `Reject_File`) .

If the contacted PoA finds acceptable the request, it sends back to the user's browser the object defined by the value of the `AURL` parameter of the request (or a default object configured in the PoA by means of `Accept_File`). Along with this object, the PoA sends the first copy of the access cookies to the browser, coded according to the received assertion and the rewrite rules defined by any `User_Data_Rewrite` and/or `Hcook_Generator` directives, and valid for the shorter of the time periods defined by `expiryTime` in the request URL or the `Max_TTL` directive.

The accept page usually contains links to the contacted sites (highlighted or produced by means of the objects requested to the PoAs), constituting sort of a portal to those resources. In this scenario, the accept page should contain a link to PoA0. When the user clicks on this link, since access cookies are already loaded when the requested was accepted, access is granted.

2.3 Scenario 2: Authorization through a GPoA

Let's consider the case when, once the steps described above have been successfully performed, the user direct the browser to PoA1. The behavior for PoA2 is exactly the same, since the PAPI access cookies are bound to the server and location the PoA is configured to, not to a entire domain. The access control handler at PoA1 finds that no access cookies are available, and it directs the browser to GPoA0 (according to the value of its `GPoA_URL` directive) using a URL of the form:

`http://gpoa0.target1.domain/AuthLocation?ACTION=CHECK&DATA=KEY&URL=PoA1URL`

Where `KEY` is an internal reference to the request data, stored in an internal database.

When GPoA0 receives this request, it verifies that it comes with the appropriate access cookies, so it prepares a response to PoA1 in the form of another redirection to the following URL:

`PoA1URL?ACTION=CHECKED&DATA=a1b2c3d4...`

Where `PoA1URL` is the value of the URL parameter in the first redirection, and the `DATA` field is somehow similar to the parameter that an AS send to a PoA, although it depends on whether the access cookies for GPoA0 were correctly received or were invalid (expired, corrupted, etc). In the case of invalid cookies it has the following format:

`ERROR:0:0:KEY`

In the case the cookies are valid, the format is as follows:

`userAssertion:expiryTime:currentTime:KEY`

Where `KEY` is the value of the `KEY` parameter in the first redirection (it is used as key in the data base where the original HTTP request has been stored, it will allow to continue with it in a transparent way for the user), and `userAssertion` is

derived from the contents of GPoA0 access cookies, once the applicable rewrites defined by any `GPoA_Rewrite` (if any) are done.

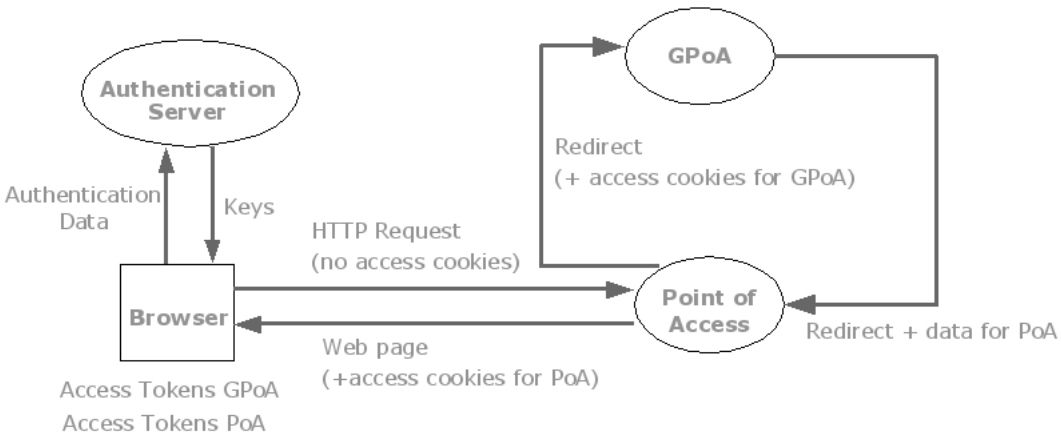

Figure 2: Interactions between a PoA and its parent GPoA

These data is signed using the private key of GPoA0, held in the file defined by its configuration directive `GPoA_Priv_Key`.

Upon receiving the redirected request back, PoA1 will:

1. Check whether a file named `_GPoA_pubkey.pem` exists in the directory defined by `Pubkeys_Path`.
2. Verify if the data sent by GPoA0 is correctly signed, by decrypting it with the key found in the previous step and checking its format.
3. Verify that the string `ERROR` has not been received in the place of the assertion about the user (that is, the access cookies for the GPoA were valid).
4. Verify that the assertion about the user has not expired (`currentTime` is inside the limits defined by the parameter `URL_Timeout`), and that the value of the request reference in `KEY` is in the internal database.
5. Check if the user assertion matches one of the filters defined by means of `PAPI_Filter` and whether if translates into an `accept` or a `reject` result.
6. If an external authorization engine has been defined by means of `SPOCP_Server`, prepare a query to the authorization server using the assertion about the user and send it to the server.

If any of these tests fails, PoA1 will reject the original user's request as if the access control phase has failed.

If all the tests pass, PoA1 will progress the original request (retrieving its parameters from the internal database) and send the corresponding answer along with the first copy of the access cookies to the browser, coded according to the received assertion and the rewrite rules defined by any `User_Data_Rewrite` and/or `Hcook_Generator` directives, and valid for the shorter of the time periods defined by `expiryTime` in the URL from GPoA0 or the `Max_TTL` directive.

2.4 Scenario 3: Attribute query from a PoA. Authentication required

Now assume that a new user tries to access directly PoA0. The mechanisms described here will be the same for PoA1 and PoA2: only an intermediate step at GPoA0 would take place, using the procedures described in the preceding section. We will concentrate in this scenario to PoA0, since the case for PoA1 and PoA2 (or at any depth in a GPoA hierarchy) is a combination of the mechanisms described here and those described in section 2.3, as we will see in the next section.

Since no valid cookies are received from the user's browser, PoA0 looks for a GPoA using its `GPoA_URL` parameter. PoA0 does not belong to any GPoA hierarchy, but it can use a special `GPoA_URL` format, that uses the `wayf:` method identifier to indicate that PoA0 must contact a a PAPI AS, and that the URL following the string `wayf:` is going to help PoA0 in finding the appropriate AS to be queried about this user. A reference of this form:

```
wayf:built-in
```

Will make the PoA software to manage by itself an internal request showing the user a list of known ASes, so one of them may be selected. In general, a reference of the form:

```
wayf:http://some.server.some.domain/wayfURL
```

Will make the PoA redirect the request to the URL identified there.

In this case, the request includes the following parameters:

`PAPIPOAREF` The reference assigned by the PoA to the request (see previous section)

`PAPIPOAURL` The URL to redirect the response from the selected AS (see previous section)

`PAPIHLI` An optional identifier intended to direct the `wayf:` handler to an AS (or set of ASes) the PoA finds appropriate to satisfy the request.

The `wayf:` handler must interact with users in any way it considers necessary to establish which is the appropriate AS to forward the authentication request to. Once it has determined the AS, it must prepare an authentication request to be sent to it. This request is very much like a request going from a PoA to its parent GPoA:

```
ASURL?ATTREQ=POAOServiceID&PAPIPOAREF=value&PAPIPOAURL=value
```

To process this request, AS0 shows the user its login page, and follows its normal authentication procedures. If the authentication fails, the reject page is sent to the user. If the authentication succeeds the authentication cookie is set (see next section), but no accept page is sent to the user, rather a redirect to the URL included in `PAPIPOAURL` is performed, with the following format:

`PAPIPOAURL?AS=AS0?ACTION=CHECKED&DATA=1a2b3c...`

Where the `DATA` field is similar to the parameter sent by a parent GPoA. If the AS could not build a correct assertion for the user the format is:

`ERROR:0:0:PAPIPOAREF`

In the case a valid assertion could be constructed, the format is as follows:

`userAssertion:expiryTime:currentTime:PAPIPOAREF`

These data is signed using the private key of the AS. It is important to note that no other PoA defined for this AS is contacted.

Upon receiving the redirected request back, PoA1 will:

1. Check whether an AS named AS0 is recognized by them, that is, it is defined by a `PAPI_AS` directive and there is a file named `AS0_pubkey.pem` in the directory defined by `Pubkeys_Path`.
2. Verify if the data sent by AS0 is correctly signed, by decrypting it with the key found in the previous step and checking its format.
3. Verify that the string `ERROR` has not been received in the place of the assertion about the user (that is, AS0 could build a valid assertion).
4. Verify that the assertion about the user has not expired (`currentTime` is inside the limits defined by the parameter `URL_Timeout`), and that the value of the request reference in `PAPIPOAREF` is in the internal database.
5. Check if the user assertion matches one of the filters defined by means of `PAPI_Filter` and whether if translates into an `accept` or a `reject` result.
6. If an external authorization engine has been defined by means of `SPOCP_Server`, prepare a query to the authorization server using the assertion about the user and send it to the server.

If any of these tests fails, PoA0 will reject the original user's request as if the access control phase has failed.

If all the tests pass, PoA0 will progress the original request (retrieving its parameters from the internal database) and send the corresponding answer along with the first copy of the access cookies to the browser, coded according to the received assertion and the rewrite rules defined by any `User_Data_Rewrite` and/or `Hcook_Generator` directives, and valid for the shorter of the time periods defined by `expiryTime` in the URL from AS0 or the `Max_TTL` directive.

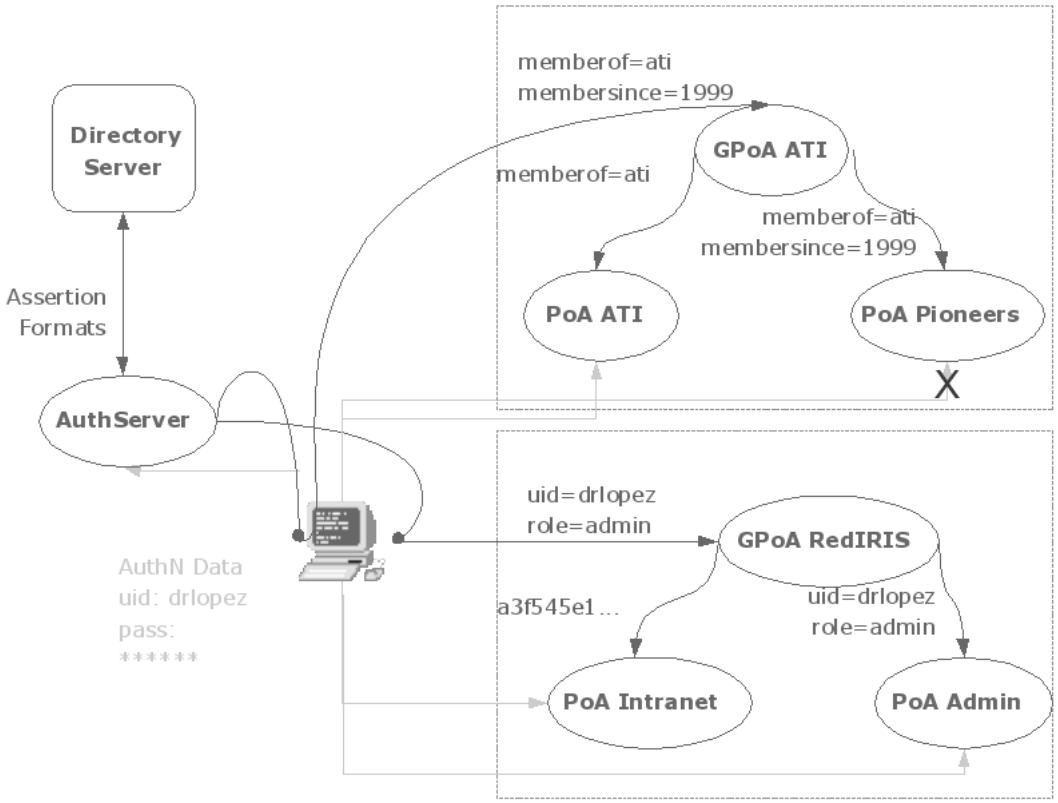

Figure 3: Attribute-based authorization using PAPI

2.5 Scenario 4: Attribute query from a PoA. Automatic reauthentication

If the user that has authenticated at AS0 as a result of requesting access to PoA0 as described in the preceding section tries now to access, say, PoA2, the point of access will detect that there are no access cookies in the request and redirect it to GPoA0, according to the procedures described in section 2.3. When this request arrives to GPoA0, it will detect also that there are no access cookies in it and will initiate a `wayf:` redirection according to the mechanisms described in 2.4.

When the request for authentication arrives at AS0, the authentication server will detect the presence of the authentication cookie (set whenever a valid authentication has been performed, as described in 2.2 and 2.4). AS0 will try to decode it using its key and check that:

1. The format of the cookie is correct, and contains the attributes described in the AS configuration

2. It has not expired.

3. It comes from the same IP address that it was issued to.

If any of these test fails, AS0 will behave as if the cookie were not set, sending the login page to the user.

If all the test pass, AS0 will issue a response to GPoA0 following the steps described in section 2.4 above. Once GPoA0 receives the response from AS0, it will prepare a response for AS1 following the mechanisms described in section 2.3.

2.6 How assertions are built by the AS

A PAPI authentication server is able to decide which assertion is going to send about which user for a certain PoA in a completely individual basis, thus providing fine-grained control over privacy for the users. Of course, it can also use system-, group-, or site-wide defaults, so AS administrators can comfortably control these settings for their systems.

Since a PAPI AS is highly modular, the way in which assertions are generated depends on the particular hooks that are used. When no other assertion format is provided, the AS uses by default the value of the configuration variable `Assertion`. The value of this variable is a string, inside of which the constructs `<papi var="VarName"/>` are substituted by the value of the variable identified by `VarName` inside the AS. If not set (and no other format has been obtained for a PoA), the assertion sent is equivalent to:

```
<papi var="PAPIuid"/>-<papi var="PAPIgid"/>
```

In the current implementation, only LDAP-based authentication supports the use of individualized assertions. This is done by means of the attribute `papiAssertion` in the LDAP class `papiSite` (describing a PoA at a certain AS) and the attribute `papiQualifiedAssertion` in the LDAP classes `papiUser` (describing a user of the AS) or `papiGroup` (describing a group of sites with common properties for the AS). To decide which assertion is going to be sent to a given site, the LDAP assertion generation functions apply the following procedure:

1. Use the format defined by the attribute `papiQualifiedAssertion` matching the site identifier defined in the `papiUser` entry.

2. If (1) is not applicable, use the format defined by the attribute `papiQualifiedAssertion` that matches the site identifier in any of the applicable `papiGroup` objects.

3. If (2) is not applicable, use the format defined by the attribute `papiAssertion` of the `papiSite` object for the corresponding site.

4. If (3) is not applicable, the value defined in the AS configuration is used (see above).