FAME-PERMIS Project Output

WORKPACKAGE 7

– Appendix A1 for the FAME-PERMIS Final Report
(Deliverable D10 version 2)

FAME Design Final Report

Aleksandra Nenadić
Ning Zhang

School of Computer Science
University of Manchester

March 2007
# Table of Contents

1. Introduction ................................................................. 2
2. The Shibboleth Infrastructure ........................................ 5
   2.1. Shibboleth Architecture ........................................... 5
   2.2. Shibboleth Architectural Components .......................... 6
   2.3. Shibboleth Components Interactions ............................ 7
3. FAME, PERMIS, and Shibboleth ....................................... 8
4. The FAME System ......................................................... 9
   4.1. FAME System Functional Overview .............................. 9
   4.2. FAME Architecture ................................................. 9
   4.3. Deriving Loa ....................................................... 11
   4.4. FAME Logic ........................................................ 12
       4.4.1. FAME Tokens .................................................. 13
       4.4.2. FAME Authentication Scenarios .......................... 14
       4.4.3. FAME Component Logic and Flow ......................... 16
       4.4.4. FAME Keys ..................................................... 17
       4.4.5. FAME Token Structure ..................................... 18
       4.4.6. FAME Token Verification ................................. 20
       4.4.7. FAME Timeouts .............................................. 21
5. FAME Configuration and Installation ............................... 21
   5.1. Apache Configuration ............................................. 22
   5.2. Database Configuration .......................................... 28
   5.3. Authentication Server(s) Configuration ...................... 29
6. FAME and Shibboleth Integration ...................................... 31
   6.1. Extending the Shibboleth’s LDAP Directory Schema ........... 31
   6.2. Configuring Attribute Definitions and JNDI Data Connectors .. 32
   6.3. Configuring Shibboleth IdP’s Attribute Release Policy ....... 33
The FAME System Design Final Report

1 Introduction

This document describes the design and implementation of the FAME (Flexible Access Middleware Extension) system’s architecture and components. The work is part of the FAME-PERMIS project aimed at providing a multi-level and risk-based user identification/authentication service in the Shibboleth\(^1\) infrastructure that has been endorsed by the JISC (Joint Information Systems Committee) as the next generation authentication and authorisation infrastructure for the UK Higher Education community. The FAME system provides authentication services to Shibboleth Identity Providers. It derives a Level of authentication Assurance (LoA) based upon the authentication strength of the authentication token and protocol used by a user in an authentication instance, and passes it over, via Shibboleth message passing protocol, to the PERMIS\(^2\) authorisation decision engine for achieving LoA based fine-grained access control.

Shibboleth technology defines open-source protocols and entities, including Identity Providers (IdPs), Service Providers (SPs), and Where Are You From (WAYF) service, to support locally authenticated users to access remote resources provided by various institutions (SPs) inside a federation. Shibboleth itself does not provide authentication and nor authorisation services – rather it leaves the task and the means of identifying a user to the user’s home institution (IdP) and authorisation decision making to the SP which manages the resource the user is making access to. Instead, it defines a set of protocols for the secure assertion of the user’s attributes between an IdP and a SP. By using the user’s attributes from the assertion made by the IdP, the SP makes its authorisation decisions. Shibboleth assumes that the IdPs and SPs form a federation which has pre-defined trust relationships. So, an SP trusts an IdP to authenticate its users properly and an IdP trusts an SP to make correct access control decisions for its users. In addition, Shibboleth IdPs can allow their users to choose what attributes can be released to a specific SP. For example, a user may choose not to release his/her true identity information to an SP, thus preserving his/her identity privacy.

Shibboleth does not mandate which authentication system an IdP should use – it is left to IdP’s local security policies and requirements. Different authentication methods, tokens, and protocols provide different levels of authentication assurance\(^3\). For instance, IP address-based authentication and authorisation services would grant the access privilege to anybody who has access to a machine with a correct IP address. Authentication via username/password establishes the identity of a user through proving the knowledge of an authorised username/password pair. A smart-card based authentication method authenticates a user provided that the user possesses a hard cryptographic token and also can demonstrate the knowledge of a secret (or a PIN) used to lock/unlock this token. Clearly, the IP-based authentication method provides the weakest, whereas the smart-card based method provides the strongest level of authentication assurance among the three methods.

---

\(^1\) The Shibboleth project, http://shibboleth.internet2.edu/.


\(^3\) http://csrc.nist.gov/publications/nistpubs/800-63/SP800-63V1_0_2.pdf.
Access privileges granted to the user should be linked to the assurance level of the authentication token/protocol used in the particular authentication instance. Such a linkage is necessary for the provision of fine-grained access control and privilege allocation in Grid environments in which the same or different applications may have dissimilar authentication requirements as dictated by varying levels of resource sensitivity and access mode towards different groups of users. For example, services like e-journal subscription or e-learning services may have a relatively low sensitivity level and therefore can be accessible to everybody who can be identified by the IP address of his/her machine. Tutors/examiners may need to use a stronger form of authentication than that used by students in order to access, say, exam papers, as the former bear more responsibility with regard to the confidentiality and integrity of the data resource. Similarly, in a Health Grid context, electronic patient records (EPRs) and electronic health records (EHRs) are shared among GPs, clinicians, and clinical and biomedical researchers across different institutions and organisations. EPRs/EHRs have high levels of privacy requirements due to legal and ethical reasons. Therefore, it is usually expected that EPRs/EHRs are de-personalised and sensitive information that can be used to identify the owner of a record are removed, before being released to entities outside hospital premises or before the researchers are allowed to access them. Password-based authentication methods may be sufficient to identify researchers when accessing these de-personalised records. However, the suppliers of the records, e.g. GPs and hospitals, should use a stronger form of authenticators when uploading new records into the de-personalised data repository due to privacy and accountability concerns. Therefore, there is clearly a need for a fine-grained access control framework to satisfy the complex VO access control requirements, and one important element of the access control decision making is the authentication strength of the authenticator used by the user.

The FAME-PERMIS project is aimed at realising this vision of sensitivity level or risk level based authentication and authorisation. With this approach, an SP may specify a minimum LoA depending upon the resource sensitivity and/or risk levels, and require that the access is granted only if the LoA derived from an authentication instance satisfies the minimum LoA.

The FAME system is designed to protect Web-based resources and applications (including the Shibboleth) with multiple authentication services. FAME can easily be integrated with any authentication services with a Web-based front-end (e.g. Kerberos, NIS (Network Information Service), and authentication systems that use LDAP (Lightweight Directory Access Protocol) or Mysql, etc.), and supports the use of various authentication tokens including username/password pairs, soft certificate tokens, hard certificate tokens stored in smart-cards and IP addresses. In the Shibboleth infrastructure, FAME plays the role of an institution’s (i.e. IdP’s) local authentication system. Based upon the token used in an authentication instance, FAME derives a level of (authentication) assurance (LoA). The derived LoA is then passed in the form of an attribute to the resource provider (i.e. SP) via the Shibboleth security assertion protocol, along with other information (attributes) about the user that the institution/user is willing to release. The SP subsequently feeds the LoA into an authorisation decision engine, namely PERMIS, to achieve fine-grained access control over its resources. In other words, resources on the SP’s side will now be additionally protected based on the strength of the authentication method the used has

---

4 www.fame-permis.org/.
used to prove his identity. That is, PERMIS’ authorisation decisions will be based on
the following tuple: (Subject, Target, Action, LoA), where Subject is the user
accessing the resource, Target is what the user is accessing, and Action describes what
action subject is allowed to perform on the target, and LoA is the Level of Assurance
the user has achieved when authenticating himself for the current Shibboleth session.
In addition, FAME has built-in support for single-sign-on (SSO), a facility which
enables a user to authenticate once and gain access resources provided by multiple
sites without re-authentication.

The FAME and PERMIS components are both integrated into the Shibboleth
infrastructure. The remaining part of this report describes the FAME-Shibboleth-
PERMIS integration, its architecture and how the FAME component interopera-
tes with the PERMIS component through the Shibboleth infrastructure, as well as the
detailed design of the FAME system. In detail, Section 2 describes the architecture
and protocols of Shibboleth, and Section 3 explains how the two set of components,
FAME and PERMIS, fit inside the Shibboleth framework. Section 4 is devoted to the
detailed designs of the FAME architecture and its components. Installation
instructions for the FAME system are available as separate document, ‘FAME System
User Guide’.

2 The Shibboleth Infrastructure

2.1 Shibboleth Architecture

The Shibboleth defines two sets of architectural entities, a set for the origin site (also
referred to as an Identity Provider or IdP) and a set for a target site (resource/service
providers or SPs). The IdP and SP sites form a federation, and, in the Shibboleth
infrastructure, different sites of the same federation are assumed to trust each other. In
other words, an SP an IdP to identify its users and provide correct attributes about the
users. Based on the attributes that are asserted by the IdP, the SP makes a decision as
whether the user should be granted the access to the requested resource.

The user’s IdP is responsible for identifying and authenticating the user, and
providing signed attribute assertions about the user to a target site from which the
user has requested resources. Shibboleth does not mandate how user authentication
should be performed at the IdP – the only requirement is that the IdP runs a Web-
based authentication service. Authorisation is performed at an SP (e.g. by using the
PERMIS access control decision engine) and is based on the attributes asserted by the
IdP. The access control mechanism matches the obtained attributes against the policy
rules associated with the requested resource in order to determine whether the
requester is permitted to access it. Similarly, Shibboleth does not mandate which
authorisation engine should be used by the target site. In other words, in the
Shibboleth infrastructure, the authentication and authentication decisions are left to
(and managed locally at) the IdP and the SP, respectively. The only requirement is
that the Identity Provider and the Service Provider agree and understand the meaning
of the attributes and implement the Shibboleth protocol for their request and release.
In this way, Shibboleth detaches the management of users at cooperating institutions
so that only IdPs are involved with verifying the identity of users belonging to their
institution.
Shibboleth has an embedded mechanism to protect users’ identity privacy, i.e. to prevent the revelation of the user’s identity to the SP: attributes released from an IdP to an SP need not contain information about the user’s identity, but rather a user’s pseudonym, e.g. an assertion on a group that the user belongs to (such as student or member of staff). In this way, the user’s identity privacy is preserved. Furthermore, a user would have the option of specifying exactly what attributes should be released to a specific SP.

2.2 Shibboleth Architectural Components

The Shibboleth architecture along with its architectural components is illustrated in Figure 1. On the IdP’s side, Shibboleth has four components: the Attribute Authority (AA), the Handle Service (HS), a directory service (such as LDAP) containing users’ attributes and a Local Authentication System (LAS). The latter two components are not provided by Shibboleth. The HS is responsible for determining the identity of the user by interacting with the LAS and creating an opaque handle (i.e. a reference number) for the user. The word ‘opaque’ means that no entity other that the AA and the HS should learn anything about the user from examining the handle alone. The handle will be used as an index number for the Shibboleth Attribute Requester (SHAR) from the target site to later fetch the user’s attributes from the AA. Shibboleth does not specify how the HS knows the identity of the user, but it is assumed that the user is authenticated to the LAS prior to being allowed to hit the HS. The AA manages all the users’ attributes at the IdP.

![Shibboleth Architecture Diagram](image_url)

Figure 1: The Shibboleth Architecture

On the SP’s side, Shibboleth has three components: the Shibboleth Indexical Reference Establisher (SHIRE), the Shibboleth Attribute Requester (SHAR), and the Resource Manager (RM). SHIRE monitors the users’ requests for the resource on the target: if a user has not been authenticated by the IdP yet (as determined by the absence of the handle), the user will be asked to go back to his IdP and do so. Once the user is authenticated and re-directed back to the SP with the handle obtained from his IdP, SHAR uses the handle to ask for the user’s attributes from the IdP. RM makes access control decisions by using an authorisation engine, once user’s attributes are fetched from the IdP.
In addition, there is a WAYF (Where Are You From) service in the architecture. This service is used by an SP to find out the address of an IdP for a particular user. The WAYF service may determine the address of an IdP by allowing the user to select his/her origin institution from a list of institutions that have been registered with the federation. The WAYF service may be provided by a third party or directly by a SP.

2.3 Shibboleth Component Interactions

Figure 2 shows the interactions between the origin (identity provider) and the target (resource/service provider). The dashed lines and boxes represent optional interactions. From the Figure, it can be seen that a user initiates a request through his/her browser for a Shibboleth-protected resource provided by an SP (e.g. www.target-site/resource as shown in Figure 1). The SHIRE component at the SP intercepts the request to ask for a handle for the user (if there is none already contained in the request). For doing so, the SHIRE creates an Attribute Query Handle (AQH) message which the user’s browser carries to the HS located on the user’s IdP. To determine the address of the user’s HS, the WAYF service can be used. This service may allow the user to select his origin institution (containing the address of the origin’s HS) from a list of institutions that have been registered with the federation. Alternatively, if the SHIRE already knows the address of the user’s origin HS, the WAYF service can be skipped over. The AQH message also contains the SHIRE’s handle acceptance URL (i.e. the address to which the HS is to return the handle) and the originally requested target URL that should be re-requested upon successful authentication.

The HS at the origin is protected by the origin’s LAS (Local Authentication System). So, before the HS is invoked by the browser carrying the AQH message, the user
needs to authenticate him/herself to the LAS. Upon successful authentication, the HS creates a handle (i.e. a reference number) for the user. The handle package (called AQH presentation) is passed back to the SHIRE via the HTTP POST method (by including it in a form that gets posted to the SHIRE’s handle acceptance URL). The handle package contains the handle for the user and the address of the AA that should be queried further for the user’s attributes. The actual handle is in the SAML\(^5\) format and is digitally signed (i.e. asserted) by the HS. Now when the user hits the SHIRE again, the SHIRE detects that the user browser contains the handle, and upon the verification of the authenticity of the handle (i.e. the verification of the HS’s signature in AQH presentation), the SHIRE passes it to the SHAR.

Upon the receipt of the handle, the SHAR sends an Attribute Query Message (AQM) containing the handle to the user origin’s AA to ask for the user’s attributes. The AA determines the user’s identity using the reference number contained in the handle, retrieves the user’s attributes related to the requested target from an attribute directory (e.g. LDAP), and constructs a reply, called an Attribute Response Message (ARM) containing these attributes. Both AQM and ARM are SAML messages. Based on the user’s attributes in the assertions, the decision is then made by the RM as whether to return the requested resource or an (HTTP) error to the user.

3. **FAME, PERMIS and Shibboleth Integration**

This section explains how FAME and PERMIS components fit into the Shibboleth infrastructure (see Figure 3 for the illustration).

---

**Figure 3: FAME-Shibboleth-PERMIS integration**

PERMIS is integrated within Shibboleth infrastructure to provide SPs with a policy-based access control decision engine to enable controlled use of their resources. The

---

\(^5\) SAML (Security Assertions Markup Language) is an emerging OASIS standard used by Shibboleth for its attribute assertions (see http://www.oasis-open.org).
attributes obtained by the target’s SHAR from the origin’s AA are fed into PERMIS to make access-control decisions. One of the attributes passed from the origin to the target is LoA, which is derived by the FAME component upon the authentication method/token used by the user in the identification process.

FAME, on the other hand, is a part of Shibboleth IdPs, and plays the role of the Local Authentication System (LAS) as shown in Figure 3. It is designed to offer users a variety of authentication methods supported by the origin, calculate the LoA based upon the authentication method/token used, and include this LoA together with the user’s other attributes ready to be fetched by Shibboleth targets.

4 The FAME System

4.1 FAME System Functional Overview

As a risk-based authentication service, FAME ensures that only the users, who have been authenticated to the authentication service satisfying the minimum LoA, are allowed to hit the HS. The functionalities of the FAME are summarised as follows:

1. Facilitate controlled access to the Shibboleth’s HS;
2. Allow a user to choose among a list of authentication methods (i.e. Authentication Servers) supported/provided by the IdP;
3. Invoke the appropriate Authentication Server (AS) based upon the user’s choice and/or as required by the minimum LoA;
4. Calculate the LoA based on the authentication strength of the selected authentication service;
5. Add the LoA to the set of user’s attributes to be passed back to the Shibboleth target;
6. Set the REMOTE_USER environment variable to pass the identity of the authenticated user to the HS;
7. Provide the Single Sign-On (SSO) functionality. Instead of invoking an authentication process each time when the user’s requests are directed to the Shibboleth’s HS, FAME only authenticates the user for the first time in a session. Upon successful authentication, the user will be issued with an “SSO token” (by FMAE) so that for subsequent requests in the session (and/or before the session times out), the user will not need to re-authentication him/herself. The token can be reused every time when an access request is made by the user to the HS, until the token expires (or the user’s current SSO session ends). In other words, re-authentication of the user (provided that the user does not wish to do so) is not necessary within the validity period of an SSO token.

4.2 FAME Architecture

The FAME system architecture consists of the following components (as shown in Figure 4): a User-Agent (UA), two FAME internal components, Shibboleth’s HS running inside an Apache Web Server, and a set of Authentication Services/servers (ASs) run by the IdP. The User-Agent (UA) is a Web browser that supports the use of cookies. The two FAME components are:

1. FAME SSO Checker (F-SSO), and
(2) FAME Login Server (F-LS).

These two components jointly provide the functionality defined for FAME, i.e. LoA linked authentication, and SSO. The Authentication Servers are Web-based authentication services external to the FAME components. These include any existing authentication services that can be invoked via a Web-based interface. Examples of such services are Kerberos, NIS, authentication via LDAP directory and MySQL database, SSL authentication with PKI soft and hard tokens. FAME does not re-implement all these existing authentication services, but rather integrates itself with them and invokes them as necessary. Furthermore, FAME is designed in such a way that it will support the use of any Web-based authentication system (including existing standard or custom built, as well as emerging ones).

![FAME architecture diagram](image)

**Figure 4: The FAME architecture**

**FAME SSO Checker (F-SSO)**

The FAME SSO Checker (F-SSO) plays the role of a gate-keeper by controlling the access to the Shibboleth’s HS (Handle Service). If a user has not been authenticated in the current session (as determined by the absence of an sso cookie in the user’s Web browser) the user will be redirected to the FAME Login Server (F-LS) and be forced to go through an authentication process. Otherwise, if the user has already been authenticated and issued with an sso cookie by the F-LS previously during this session, the access to the HS will be granted by the F-SSO and the current Shibboleth session will continue without re-authentication provided that the cookie is still valid.

**FAME Login Server (F-LS)**

The role of the FAME Login Server (F-LS) is to receive users directed from the F-SSO and redirect them to an Authentication Server (AS) supported by the site. If a site has two or more ASes supported (this may be determined by the origin’s authentication policy and needs), then the FAME interface window will allow a user to choose one from the list of the supported ASes (that satisfy the specified minimum LoA), and the F-LS will redirect the user to the AS of his/her choice. Once the authentication is performed and successful, the user is re-directed back to the F-LS by the AS. The AS will also pass back the userID to the F-LS. Note that the userID varies with different authentication methods used. For instance, with username-password based authentication, the userID is the user’s username; with Kerberos, the userID is the principal’s name in the form of <Fully Qualified Domain
Name>@<Realm Name>;< for LDAP authentication, the userID is user’s DN (Distinguished Name); for certificate based authentication, the userID a combination of C=<Country>, ST=<State>, L=<Locality>, O=<Organisation>, OU=<Organisational Unit>, CN=<Common Name> extracted from the user’s certificate; etc. Also note that one user can have multiple userIDs as he/she may register with multiple authentication services available on on the IdP site. However, at any one time, the user can only choose one authentication method.

Once a user has been successfully authenticated by, and re-directed from, the AS, the user is issued with an sso cookie by the F-LS. The cookie is scoped for the use by the F-SSO and contains the userID received from the AS and the LoA derived by F-LS based upon the authentication method/token used by the user. The SSO cookie also contains other information; detailed descriptions of the contents of the SSO cookie are given in Section 4.2.1. After the sso cookie is generated, the F-LS re-directs the user back to his originally requested URL (i.e. the url of the Shibboleth’s HS). This time, when the user tries to access the HS, the request will again be intercepted by the F-SSO. However, as now the user’s request contains the sso cookie created by the F-LS, the F-SSO will let the user through to the HS. In addition, the F-SSO will pass to the HS the userID (contained in the sso cookie) through the environment variable REMOTE_USER in order to supply the HS with the identity of the browser user. This information will be used by the HS to obtain a handle for the user. On the other hand, if the authentication with the AS is not successful, it is up to the AS to display an error message to the user and the user will not be re-directed to the HS.

All the interactions between the UA and the F-SSO and between the UA and the F-LS are SSL-protected to prevent cookies from being disclosed to an eavesdropper that could use them to impersonate the user and to gain unauthorised access to the HS. All the interactions between the UA and the AS are SSL-protected to prevent disclosure of passwords or other sensitive credentials. In the following, we give a more in-depth coverage of the FAME logic and authentication scenarios.

4.3 Deriving LoA

According to the NIST Electronic Authentication Guideline (NIST Publication 800-63⁶), there are four levels of authentication - Level 1 to Level 4. Level 1 provides the lowest level of assurance and Level 4 the highest. These levels are defined in terms of the likely consequences of an authentication error. As these errors become more serious, the required level of assurance increases. A short summary of the technical requirements for each of the four levels is provided below.

Level 1 – Successful authentication requires the claimant to prove that he/she controls the authentication token through a secure authentication protocol. Any of the authentication tokens of Level 2, 3, or 4 are allowed, as well as PINs. Plaintext tokens (e.g. passwords) are never transmitted across the network. However, this level does not require the use of cryptographic protocols that block off-line attacks. For example, password challenge-response protocols are allowed at this level, and an eavesdropper, having intercepted such a protocol exchange, can launch an off-line dictionary attack in order to discover the password. Therefore, there is no requirement at this level to

---

use FIPS (Federal Information Processing Standards) approved cryptographic techniques.

**Level 2** - Successful authentication requires the claimant to prove that he/she controls the authentication token through a secure authentication protocol. Any of the authentication tokens of Level 3, or 4 can be used, as well as passwords. The use of cryptographic protocols that can prevent off-line, replay and on-line guessing attacks is required. FIPS approved cryptographic techniques are required.

**Level 3** - Successful authentication requires the claimant to prove that he/she controls the authentication token (i.e. to prove a possession of a key or a password) through a secure authentication protocol. Three kinds of tokens may be used: cryptographic (soft and hard) tokens, one-time password device tokens and password tokens used in zero-knowledge password protocols. The use of cryptographic protocols that can prevent off-line, replay, on-line guessing, verifier impersonation and man-in-the-middle attacks is required. FIPS approved cryptographic techniques are required. All sensitive data transferred is cryptographically authenticated and, optionally, encrypted under keys derived from the authentication process.

**Level 4** - Successful authentication requires the claimant to prove that he/she controls the authentication token (i.e. to prove a possession of a key or a password) through a secure authentication protocol. This level is similar to Level 3, except that only hard cryptographic tokens are allowed. These tokens are hardware (physical) devices that cannot be easily copied and which must be unlocked with a password or a piece of biometric, and, thus, provide two-factor authentication. Either public or symmetric key technology may be used and FIPS approved cryptographic techniques are required.

<table>
<thead>
<tr>
<th>Table 1: Authentication token types vs LoAs⁷</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hard crypto token</td>
</tr>
<tr>
<td>One-time password device</td>
</tr>
<tr>
<td>Soft crypto token</td>
</tr>
<tr>
<td>Passwords &amp; PINs</td>
</tr>
</tbody>
</table>

In the FAME system, it is the responsibility of the Authentication Server (AS) to provide the FAME administrator with the correct LoA, based on the cryptographic protocols and tokens used in the authentication process provided by the AS. This LoA is then hard-coded into FAME by the FAME administrator, i.e. inserted into the database that holds configuration information of all ASs that the FAME system interacts with.

### 4.4 FAME Logic

The FAME authentication service makes use of several tokens and techniques to achieve SSO. The ideas are based on the SSO solutions described in the “Eagle Book”⁸, PubCookie⁹, WebAuth¹⁰ and Apache-AuthTicket¹¹.

---

⁸ The “Apache Modules with Perl and C” book by Lincoln Stein and Doug MacEachern.
4.4.1 FAME Tokens

To store and exchange information between different system components, to allow the components to mutually authenticate each other and to achieve SSO functionality, FAME makes use of several tokens, as outlined in Table 2. These tokens are transferred between components either as cookies or URL parameters. Cookies are only used for passing information between the two FAME internal components (namely, F-SSO and F-LS), while URL parameters are used to pass information between the FAME internal component, F-LS, and external ASes. A more detailed explanation of the contents of the tokens is given in Section 4.2.4.

Table 2: FAME Tokens

<table>
<thead>
<tr>
<th>Token name</th>
<th>Passed From → To</th>
<th>Token protection</th>
<th>Token description</th>
</tr>
</thead>
<tbody>
<tr>
<td>request-url cookie</td>
<td>F-SSO → F-LS</td>
<td>The token is passed only over an SSL-protected channel.</td>
<td>Generated by the F-SSO and scoped for the use by the F-LS. It contains the address of the originally requested url (of the Shibboleth’s HS), which was intercepted by the F-SSO.</td>
</tr>
<tr>
<td>sso cookie</td>
<td>F-LS → F-SSO</td>
<td>The token is passed only over an SSL-protected channel. It also contains a “cryptographic signature” created as a keyed hash of the token data with a secret key known only to the F-SSO and F-LS.</td>
<td>Generated by the F-LS and scoped for the use by the F-SSO after the user has successfully authenticated in order to implement SSO for subsequent resource accesses by the user via FAME. This token is not used by any external component.</td>
</tr>
<tr>
<td>auth-request token</td>
<td>F-LS → AS</td>
<td>Encrypted with the secret key shared between the F-LS and the AS and is only passed via an SSL-protected channel.</td>
<td>Generated by the F-LS and sent to the AS when requesting user authentication with the AS. It contains a random challenge RC that is used to authenticate the AS to the F-LS when response from the AS is received in the form of auth-reply token.</td>
</tr>
<tr>
<td>auth-reply token</td>
<td>AS → F-LS</td>
<td>Encrypted with the secret key shared between the F-LS and the AS, and only passed via an SSL-protected channel.</td>
<td>Generated by the AS once the user has been successfully authenticated and before the user is directed back to the F-LS. It contains a random challenge (that equals to the random challenge received from the auth-request token incremented by 1, i.e. RC + 1) and the userID of the authenticated user.</td>
</tr>
<tr>
<td>auth-control cookie</td>
<td>F-LS → F-LS</td>
<td>The token is passed only over an SSL-protected channel. It also contains a “cryptographic signature” created as a keyed hash of the token data with a secret key known only to the F-SSO and F-LS.</td>
<td>Generated by the F-LS and scoped for the use of the F-LS only. It contains the same random number sent to the AS in auth-request token and is used to verify the authenticity of the AS’s auth-reply token.</td>
</tr>
</tbody>
</table>

4.4.2 FAME Authentication Scenarios

In the following, we use use-case scenarios to illustrate the working of the FAME system. Steps of authentication procedures are presented together with illustrating diagrams where redirection between the components is represented using dashed lines.

**Scenario 1: Access to the Shibboleth’s HS without an sso cookie (i.e. during initial sign-on)**

A user initiates a new session to access the FAME-protected Shibboleth’s HS. In this case, no sso cookie has been generated yet by F-LS. In detail,

1. A user makes a request for accessing an SP, and the SP redirect the user to the url of the Shibboleth HS (running inside an Apache Server at the origin) that uses FAME for authentication.
2. The F-SSO intercepts the request and verifies that it is not associated with a current valid authenticated session for the requested resource. This is achieved by checking whether the request contains a valid sso cookie that has been previously issued by the F-LS. Consequently, the F-SSO generates the request-url cookie that contains the address of the originally requested resource (i.e. that of the HS) and redirects the user to the F-LS for the user to perform authentication.
3. The F-LS determines a list of the available Authentication Servers (ASes) each with an associated LoA, and sends a form page with the list back to the user.

4. The user is presented with the page containing the AS choices and chooses his/her preferred authentication method (or the method that satisfies the minimum LoA requirement).

![Figure 5: Application Access with no sso Cookie](image)
5. The user sends his choice back to the F-LS by submitting the form.
6. The F-LS redirects the user to the selected AS after generating the auth-
control cookie (containing the random challenge RC) and the auth-request
token (that also contains the random challenge passed to the AS, i.e. RC).
7. The user is presented with a login page by the AS where the user enters his/her
login credential, if authentication methods such as username/password-based
ones are chosen. However, for other authentication methods, such as SSL
client certificate authentication, the user will not be prompt for entering any
credentials. Authentication in these cases is performed without further
interactions between the user and the AS, and the user’s authentication
credential is sent to the AS automatically by the user’s browser.
8. The user’s credential is sent by the browser to the AS. If the authentication is
not successful, the user is prompted to re-enter the credential until the
authentication is completed successfully.
9. If the authentication is successful, the AS generates an auth-reply token
containing the userID of the authenticated user and the random challenge
incremented by one, i.e. RC+1, and re-directs the user back to the F-LS.
10. The F-LS verifies the auth-reply token by confirming that the random number
contained inside the token, (RC+1), is equal the random number from the
auth-control cookie incremented by 1. This verification confirms the
authenticity of the AS to the F-LS as the tokens exchanged between them (i.e.
auth-request and auth-reply tokens) are encrypted with the secret key shared
by these two components. If the verification is positive, the F-LS generates an
sso cookie for the use by the F-SSO. The F-LS then redirects the user to the
originally requested url (which was received through the request-url cookie).
11. This redirected request will cause the browser to re-request the original
resource. This time, when the request is intercepted by the F-SSO, the sso
cookie issued by the F-LS will be detected by the F-SSO.
12. The F-SSO verifies the sso cookie, and if the verification is successful the F-
SSO lets the user through to the HS.

Scenario 2: Access to the Shibboleth’s HS with sso cookie (Single Sign-On)

In this scenario, a user request, containing a valid sso for this session, is redirected to
the HS (this is because the user has already gone through the Scenario 1 mentioned
above in this session, i.e. he/she has previously visited the HS and has been
successfully authenticated to the F-LS).

![Diagram](image_url)

**Figure 6: Application Access with an sso Cookie**

1. A user’s request is redirected to the HS, that uses FAME for authentication,
but intercepted by F-SSO.
2. The F-SSO detects a valid sso cookie (which the user has obtained earlier in the session) and grants the user the access. If the sso cookie has expired or is invalid, the user will be redirected to the F-LS for re-authentication.

4.4.3 FAME Component Logic and Flow

F-SSO Logic

- Does the request have a valid sso cookie?
  - Yes: Satisfy the request and let the user through to the requested Shibboleth’s HS.
  - No: Redirect the user to the F-LS.

F-LS Logic

- Does the request have a valid request-url cookie?
  - Yes: Send a list of available ASs to the user. Has the user submitted his preferred AS?
    - Yes: F-LS generates an auth-control cookie scoped for the F-LS itself that will keep a random RC. (2) F-LS generates an auth-request token scoped for the AS containing the same random challenge RC. In addition, it also contains the return address of the F-LS, to which the user will be re-directed upon successful authentication. (3) F-LS redirects the user to the selected AS. (4) Has the user been returned from the AS with a correct auth-reply token?
      - Yes: F-LS sets the sso cookie, empties request-url and auth-control cookies and redirects the user to the originally requested address.
      - No: The AS will display the error message to the user.
  - No: The user cannot continue unless an AS is selected.
    - No: Generate an error page.

AS Logic

- Does the request have a valid auth-request token?
  - Yes: Authenticate the user. Has the user authenticated correctly?
    - Yes: Decrypt the auth-request token and retrieve the random challenge RC and the return address of the F-LS. Generate the encrypted auth-reply token containing the response to the random challenge and the authenticated user’s userID and redirect the user back to the F-LS.
    - No: Ask user to re-authenticate.
  - No: Generate an error page.

Figure 7 depicts the flow between the FAME system components.
4.4.4 FAME Keys

FAME system components use two secret (symmetrical) keys to encrypt the tokens passed between them in order to protect their confidentiality, authenticity and integrity. The internal components of the FAME system, namely F-SSO and F-LS, share and use the same secret key, called FLS_KEY, to protect cookies passed between them (namely, the request-url, auth-control and sso cookie). These two components use the same key as they are essentially part of the same Apache module. The FLS_KEY is used to protect the authenticity and integrity of the cookies by generating a “cryptographic signature” that a hash value computed using the secret key and the data in the cookie. In other words, the “cryptographic signature”, $H$, of a cookie is generated as follows:

$$H = \text{md}5(\text{FLS\_KEY}, \text{md}5(\text{FLS\_KEY}, \text{<cookie\_data>})).$$

As cookies are sent over an SSL-protected connection, so the confidentiality of the cookies is also preserved, and there is no need to double encrypt the cookie data.

Each FLS_KEY also has a version number associated with it. This allows the site administrator to issue a new secret key, FLS_KEY, periodically without invalidating valid cookies that were generated previously with a former version of the FLS_KEY, but are currently still in use. For example, the site administrator might periodically insert a new secret key into the secret key database, and erase any secret keys that are more than 2 days old. Since the cookie issued to the user contains the version
information, the authentication process will still accept the cookies that are generated by the secret key as long as the secret key is not erased from the secret key database. The Apache Server should be restarted after the key is updated, to allow the latest key to be used by the F-LS and F-SSO.

In addition, the F-LS component also shares a secret key, called FLS_AS_KEY, with each of the external Authentication Servers (ASes) that it integrates with. These keys are used to encrypt auth-request and auth-reply tokens passed between the F-LS and the ASes. These tokens are passed between the F-LS and an AS as URL parameters and therefore are visible in Web browser’s address bar. To preserve the confidentiality of the tokens (i.e. to protect against eavesdropping the token contents) and assure the authenticity of the tokens by the F-LS and the AS, these tokens are encrypted with the FLS_AS_KEY. It is the responsibility of the F-LS and AS administrator(s) to make sure that an FLS_AS_KEY is agreed between them, and is in place for the use by the pair.

4.4.5 FAME Token Structure

The structures of, and the roles played by the FAME tokens, are given below.

### Request-url cookie

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>creation_time</td>
<td>Timestamp of the cookie’s creation.</td>
</tr>
<tr>
<td>request_url</td>
<td>Originally requested address.</td>
</tr>
<tr>
<td>remote_user_ip</td>
<td>User’s IP address.</td>
</tr>
<tr>
<td>key_version</td>
<td>Version of the key used to create the hash of the cookie.</td>
</tr>
<tr>
<td>hash</td>
<td>Hash of the cookie’s concatenated fields above using secret key, FLS_KEY:</td>
</tr>
<tr>
<td></td>
<td>( \text{hash} = \text{md5}(\text{FLS_KEY}, \text{md5}(\text{FLS_KEY, creation_time, request_url, remote_user_ip, key_version})). )</td>
</tr>
</tbody>
</table>

### SSO cookie

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Role: This cookie helps with providing the SSO facility. It is created by the F-LS upon user’s successful authentication with the AS. It contains the remote user’s ID that is later passed to the Shibboleth’s HS, as well as the user’s LoA value just derived from this authentication instance (the LoA value will be used, along with the user’s other attribute values, to PERMIS via Shibboleth. The cookie also contains the user’s IP address, in order to make it more difficult for an attacker to forge the cookie. A potential attacker would have to guess the secret key and to spoof the user’s IP address in order to launch a successful attack. The cookie is created by the F-LS and used by the F-SSO to grant the access to the Shibboleth’s HS.</td>
<td></td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>Field</td>
<td>Description</td>
</tr>
<tr>
<td>---------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>creation_time</td>
<td>Timestamp of cookie creation.</td>
</tr>
<tr>
<td>remote_user</td>
<td>UserID used when user authenticated with AS.</td>
</tr>
<tr>
<td>remote_user_loa</td>
<td>LoA of the AS selected by the user for authentication.</td>
</tr>
<tr>
<td>remote_user_ip</td>
<td>User’s IP address.</td>
</tr>
<tr>
<td>key_version</td>
<td>Version of the key used to create the hash of this cookie.</td>
</tr>
<tr>
<td>hash</td>
<td>Hash of the cookie’s concatenated fields above using secret key, FLS_KEY:</td>
</tr>
<tr>
<td></td>
<td>[ hash = \text{md5}(\text{FLS_KEY}, \text{md5}( \text{FLS_KEY, creation_time}, remote_user, remote_user_loa, remote_user_ip, key_version)). ]</td>
</tr>
</tbody>
</table>

**Auth-request token**

Role: This token, passed from the F-LS to the AS as a url parameter, carries a random number challenge for the AS and the address of the F-LS, to which the user should be returned upon successful authentication.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_no</td>
<td>Random number challenge (used for authenticating the AS).</td>
</tr>
<tr>
<td>fls_address</td>
<td>Address of the F-LS to return the user upon authentication.</td>
</tr>
</tbody>
</table>

**Auth-control cookie**

Role: This is a control cookie that holds the random number challenge that is passed from the F-LS to the AS via the auth-request token. The random number is later compared to the random number received from the AS in the auth-reply token. The cookie also holds the address of the AS, the corresponding LoA, and remote user’s IP address in order to make the cookie more difficult to forge. A potential attacker would have to both guess the secret key and to spoof the user’s IP address in order to launch a successful. The cookie is created and used only by the F-LS, and emptied when the user is returned to his originally requested url.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>creation_time</td>
<td>Timestamp of cookie’s creation.</td>
</tr>
<tr>
<td>random_no</td>
<td>Random number challenge RC (used for authenticating the AS to the F-LS). This random number has the same value as the one contained in the auth-request token passed to the AS.</td>
</tr>
<tr>
<td>as_url</td>
<td>Address of the AS selected by the user for authentication.</td>
</tr>
<tr>
<td>remote_user_loa</td>
<td>LoA of the AS selected by the user for authentication.</td>
</tr>
<tr>
<td>remote_user_ip</td>
<td>User’s IP address.</td>
</tr>
<tr>
<td>key_version</td>
<td>Version of the key used to create the hash of this cookie.</td>
</tr>
<tr>
<td>hash</td>
<td>Hash of the cookie’s concatenated fields above using secret key, FLS_KEY:</td>
</tr>
<tr>
<td></td>
<td>[ hash = \text{md5}(\text{FLS_KEY}, \text{md5}( \text{FLS_KEY, creation_time, random_no, as_url, remote_user_loa, remote_user_ip, key_version}). ]</td>
</tr>
</tbody>
</table>

**Auth-reply token**

Role: This token, passed from the AS to the F-LS as a url parameter, carries the response to the random challenge received in the auth-request token and the userID of

---

12 UserID can be username with username-password based authentication, DN (Distinguished name) with certificate-based authentication, Kerberos username, etc. depending on the authentication method.
the authenticated user.

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>random_no</td>
<td>Random number from the auth-request token incremented by 1 (used for authenticating the AS to the F-LS).</td>
</tr>
<tr>
<td>remote_user</td>
<td>UserID extracted from the user’s credentials (e.g. username) when the user is being authenticated with the AS.</td>
</tr>
</tbody>
</table>

4.4.6 FAME Token Verification

Verifying an request-url cookie (performed by the F-LS)

- The cookie has correct format (i.e. no missing fields);
- The cookie is within its expiration time (set to 1 minute by default):
  \[
  \text{request-url cookie.creation_time < current_time + 1.}
  \]
- The hash value contained in the cookie is verified positively with the secret key FLS_KEY shared between the FAME internal components F-SSO and F-LS, to confirm the authenticity and integrity of the cookie. That is,
  \[
  \text{request-url cookie.hash = md5(FLS_KEY, md5( FLS_KEY, request-url cookie.creation_time, request-url cookie.request_url, connection.remote_ip, request-url cookie.key_version)).}
  \]

Verifying an sso cookie (performed by the F-SSO)

- The cookie has correct format (i.e. no missing fields);
- The cookie is within the SSO Timeout (set to 8 hours by default);
  \[
  \text{sso cookie.creation_time < current_time + 480.}
  \]
- The hash value contained in the cookie is verified positively with the secret key FLS_KEY shared between the FAME internal components F-SSO and F-LS, to confirm the authenticity and integrity of the cookie. That is,
  \[
  \text{sso cookie.hash = md5(FLS_KEY, md5( FLS_KEY, sso cookie.creation_time, sso cookie.remote_user, sso cookie.remote_user_loa, connection.remote_ip, sso cookie.key_version)).}
  \]

Verifying an auth-request token (performed by the AS)

- The AS can decrypt the token with the secret key shared between the AS and the F-LS and the obtained token has correct format (i.e. no missing fields). This token is sent from the F-LS to the AS and we do not require any additional verification here as the AS should perform authentication whenever it receives an authentication request from the F-LS.

Verifying an auth-control cookie (performed by the F-LS)

- The cookie has correct format (i.e. no missing fields);
- The cookie is received within the Authentication Timeout (set to 1 minute by default);
auth-control cookie.creation_time < current_time + 1.

- The hash value contained in the cookie is verified positively with the secret key FLS_KEY used for its creation by the F-LS, to confirm the authenticity and integrity of the cookie. That is,

  \[ auth-control \text{ cookie}.hash = \text{md5}(\text{FLS\_KEY}, \text{md5}(\text{FLS\_KEY}, \text{auth-control} \text{ cookie}.creation\_time, \text{auth-control} \text{ cookie}.random\_no, \text{auth-control} \text{ cookie}.as\_url, \text{auth-control} \text{ cookie}.remote\_user\_loa, \text{connection}.remote\_ip, \text{auth-control} \text{ cookie}.key\_version)) \]

Verifying an auth-reply token (performed by the F-LS)

- The AS can decrypt the token with the secret key shared between the AS and the F-LS and the obtained token has correct format (i.e. no missing fields). The token is received within the specified authentication timeout, i.e. within the time that is left to the user to perform the authentication and return to the F-LS from the AS (which is set to 1 minute by default);

  \[ auth-reply \text{ cookie}.creation\_time < current\_time + 1. \]

- The random number contained in the token is equal to the random number from the auth-control cookie incremented by 1. That is,

  \[ auth-reply \text{ cookie}.random\_no = auth-control \text{ cookie}.random\_no + 1. \]

4.4.7 FAME Timeouts

- The Authentication Timeout indicates the time period during which the auth-control cookie and auth-request token are created by the F-LS, the user is redirected to the AS for authentication, and the user is returned back from the AS to the F-LS after the authentication. In other words, it is the time duration given to the user for her/him to successfully complete the authentication. It is calculated as the difference of the auth-control cookie’s creation_time and the time when the auth-reply token is received. The default setting is 1 minute, but this can be changed via FameAuthTimeout directive (see Section 5.1 for details).

- The SSO Timeout specifies the time interval upon the expiration of which the user will be prompted for re-authentication. It is the time interval within which the sso cookie is considered as non-expired. It is computed using the creation_time field of the sso cookie. The default setting is 8 hours, but this can be changed via FameSSOTimeout directive (see Section 5.1 for details).

5 FAME Configuration and Installation

When configuring the FAME system, it is strongly advised that you read these FAME system configuration instructions in conjunction with the “FAME System User Guide”, which is available as a separate document.

The FAME system internal components are implemented as an Apache mod_perl module called Fame.pm. The module is created under the MyApache2:: namespace, thus its full name is MyApache2::Fame.pm. Configuration data for the FAME system
is kept in a relational database. The following tasks have to be done in order to configure the FAME system:

1. Configure Apache2 Server with mod_perl module;
2. Create the FAME database for storing secret keys and configuration information about ASs and their LoAs, and information about FAME users;
3. Set up the Authentication Server(s) to work with FAME.

5.1 Apache Configuration

The module has been developed for and tested with Apache2. It is assumed that an Apache Server with a support for Perl (via mod_perl) has been previously configured and that the Shibboleth IdP v1.3 has already been configured to run with the Apache Server.

In order to configure the FAME module (Fame.pm) to work with the Apache Server, the following three blocks have to be configured in Apache’s configuration file (httpd.conf). The first block configures the F-SSO component, i.e. to set the F-SSO as the interceptor of any attempted requests for access to the url of the Shibboleth’s HS. By doing so, the F-SSO is effectively configured as access control that protects the Shibboleth’s HS and the Apache authentication type for this location is set to ‘Fame’. The following gives an exemplar setting of such a block (and should be inserted anywhere in the main section of httpd.conf):

```html
<Location /shibboleth/HS>
    AuthType Fame
    AuthName “Fame Authentication Service”
    PerlAuthenHandler Apache2::Fame::sso_checker
    require valid-user
</Location>
```

The above block specifies the Apache2::Fame.pm module’s sso_checker() routine to handle the user authentication for all requests for the Shibboleth HS’s url (i.e. location /shibboleth/HS) on the Apache Server.

The second block that has to be configured specifies the location and settings for the F-LS component that handles the tasks of user authentication and creation of SSO cookies. It should be inserted anywhere in the main section of httpd.conf and should look similar to this:

```html
<Location /fame_login_server>
    SetHandler perl-script
    PerlResponseHandler Apache2::Fame::login_server
</Location>
```

The third block defines the address of the FAME logout handler. If, for any reason, the user wishes to terminate his current SSO session without closing his Web browser, he may do so by clicking the “logout” button on the main FAME page. This will redirect the user to the FAME logout handler, which is configured as below.
The *Fame.pm* module can be configured via a number of configuration parameters. They can either be set up in the Apache-Perl start-up script (*startup.pl*) or in Apache’s configuration file, *httpd.conf*. If you are configuring the *Fame.pm* module via the start-up script, it should include a block look like the following:

```
# startup.pl
...
# Load FAME module
use MyApache2::Fame ();

# Configuration items for the FAME module
Apache2::ServerUtil->server->push_handlers(PerlChildInitHandler => \&MyApache2::Fame::configure(
    'FameLoginServer' => '/fame_login_server',
    'FameLogoutHandler' => '/fame_logout',
    'FameAuthTimeout' => 480,
    'FameSSOTimeout' => 1,
    'FameDB' =>
        'dbi:mysql:database=fls;host=localhost;port=3306',
    'FameDBUser' => 'flsuser',
    'FameDBPassword' => 'flspassword',
    'FameSecretsTable' => 'secrets:secret_key:date_time',
    'FameASTable' =>
        'authservers:url:auth_type:saml_auth_id:loa:secret_key',
    'FameUsersTable' =>
        'fameusers:ldap_id:ldap_attribute:alternative_id',
    'FameShibLDAPServer' => 'localhost',
    'FameShibLDAPPort' => '389',
    'FameShibLDAPDN' => 'cn=Manager,dc=example,dc=com',
    'FameShibLDAPPassword' => 'your_secret',
    'FameShibLDAPBaseDN' => 'ou=shib-users,dc=example,dc=com' ))));
```

Make sure that the path to where you have saved the *Fame.pm* module is made visible to Perl. For example, if you have saved the *MyApache2* directory containing the *Fame.pm* module in `/etc/apache2/perl`, then the following line should precede the above block, i.e.

```
# Location of the FAME module
use lib '/etc/apache2/perl';
```

Alternatively, you may configure the *Fame.pm* module in *httpd.conf* by defining the configuration parameters using the PerlSetVar directive. An example of the FAME module’s configuration using the Apache’s *httpd.conf* is given below.
# httpd.conf

# Load Fame module
PerlModule MyApache2::Fame

# Fame module configuration items
PerlSetVar FameLoginServer /fame_login_server
PerlSetVar FameLogoutHandler /fame_logout
PerlSetVar FameDB dbi:mysql:database=fls;host=localhost;port=3306
PerlSetVar FameAuthTimeout 1
PerlSetVar FameSSOTimeout 480
PerlSetVar FameDBuser flsuser
PerlSetVar FameDBPassword flspassword
PerlSetVar FameSecretsTable secrets:secret_key:date_time
PerlSetVar FameASTable authservers:url:auth_type:loa:secret_key
PerlSetVar FameUsersTable fameusers:ldap_id:ldap_attribute:alternative_id
PerlSetVar FameShibLDAPServer localhost
PerlSetVar FameShibLDAPPort 389
PerlSetVar FameShibLDAPDN cn=Manager,dc=example,dc=com
PerlSetVar FameShibLDAPPASSWORD your_secret
PerlSetVar FameShibLDAPBaseDN ou=shib-users,dc=example,dc=com

# Shibboleth’s HS protected by the F-SSO
<Location /shibboleth/HS>
   PerlAuthenHandler Apache2::Fame::sso_checker
   AuthType MyApache2::Fame
   AuthName Fame
   require valid-user
</Location>

# Fame Login Server
<Location /fame_login_server>
   PerlResponseHandler Apache2::Fame::login_server
</Location>

# Fame Logout Handler
<Location /fame_logout>
   PerlResponseHandler Apache2::Fame::logout
</Location>

Whichever of the two configuration methods you select to use, remember that the following precedence rules apply:

- If a directive specified in both httpd.conf and start-up script startup.pl, the value from the httpd.conf will be used and will override any other value.
- Otherwise, if a directive is not specified either in httpd.conf or in startup.pl, the default value will be used.

The following is a list of the FAME module’s valid configuration parameters (i.e. directives).

**FameLoginServer**

This parameter specifies the relative url of the Fame Login Server (F-LS), and corresponds to the second <Location> block specified in the httpd.conf above. This directive tells the F-SSO where to redirect the user if he/she has not been authenticated yet (detected by the absence of the sso cookie).
Default value: /fame_login_server.

**FameLogoutHandler**

This parameter specifies the relative url of the Fame Logout Handler. It is used to enable the users to logout and clear all FAME-set cookies, i.e. to reset the current SSO session.

Default value: /fame_logout.

**FameAuthTimeout**

This parameter specifies the number of minutes that the user is given to successfully complete the authentication. It is computed as a difference between the creation time of an auth-control cookie (by F-LS) and the receipt of the auth-reply token (by the F-LS from the Authentication Server). If this timeout is expired, the user will be forced to log in (authenticate) again.

Default value: 1 minute.

**FameSSOTimeout**

This parameter specifies the number of minutes before the sso cookie is considered expired, i.e. the duration of the user’s SSO session. After that time, the user will be forced to re-authenticate.

Default value: 480 minutes (8 hours).

**FameDB**

This parameter specifies the url string for the Perl DBI (DataBase Interface) to use when making the connection to the FAME database. FAME has been developed using Mysql database, but a number of other databases can be used with Perl DBI (see http://dbi.perl.org for details) and configured to work with FAME.

Format: 
<perl_database_interface>:<database_type>:database=<database_name>;host=<host_name>;port=<port_number>.

Default value: dbi:mysql:database=fls;host=localhost;port=3306.
If the host is ‘localhost’ and the port is ‘3306’, they can be omitted from the FameDB string.

**FameDBUser**

This parameter specifies the username to use when connecting to the FAME database above. The specified user must have read privileges for the FAME database.

Default value: flsuser.

**FameDBPassword**

This parameter specifies the password to use when connecting to the FAME database.

Default value: flspassword.
FameSecretsTable
This parameter specifies the name of a table containing secret keys used by the F-LS and F-SSO, as well as the names of the two columns of this table used for the secret key itself and the secret key version. Typically, a timestamp is used as the secret key version number indicating the time when the key is inserted into the database.

Format:
<table_name>:<secret_key_column_name>:<secret_key_version_column_name>.

Default value: secrets:secret_key:date_time.

FameASTable
This parameter specifies the name for the table containing information about Authentication Servers, and the names of the five columns of this table that are used by FAME. The five columns are: (1) the url of the Authentication Server, (2) the authentication type that the Authentication Server provides, (3) the unique URN for the authentication method as defined by SAML1.1 (this is what is passed to the SP at the moment. In a future version of SAML, the exact authentication method may be passed to the SP in addition to LoA), (4) the LoA of the Authentication Server, and (5) the secret key shared between the AS and F-LS (in Base64 format).

Format:
<table_name>:<url_column_name>:<auth_type_column_name>:<saml_auth_id>:<loa_column_name>:<secret_key_column_name>.


FameUsersTable
This parameter specifies the name of the table containing mapping between the FAME users’ ids registered with Authentication Servers, and their corresponding DNs used in the Shibboleth LDAP directory. Each FAME user has a unique entry in the Shibboleth’s LDAP directory, identified by his LDAP DN, where his current LoA value is stored and picked up by Shibboleth. However, each FAME user may have several identities each registered with a distinctive Authentication Server, e.g. in cases when their IdP supports several methods of authentication, but none of which uses LDAP to store users’ credentials. For this reason, we have to map these different identities of the same user to the user’s identity in the Shibboleth’s LDAP directory. FameUsersTable contains such mappings; it consists of three columns: (1) the user’s LDAP id (part of the user’s LDAP DN), (2) the name of the LDAP attribute which keeps the user’s id (e.g. uid or cn), and (3) the user’s alternative id (e.g. Kerberos principal name such as kerbuser@CS.MAN.AC.UK, or subject of the public key certificate such as C=GB/ST=Lancashire/L=Manchester/O=University of Manchester/OU=School of Computer Science/CN=Alex
Nenadic/emailAddress=anenadic@cs.man.ac.uk, or any other user’s alternative username).

Format:
<table_name>:<ldap_id_column_name>:<ldap_attribute_column_name>:<alternative_id>.

Default value:
fameusers:ldap_id:ldap_attribute:alternative_id.

Note that, even if the IdP utilises LDAP for user authentication, all LDAP users must be inserted in the FameUsersTable. In this case, the <ldap_id> and <alternative_id> fields would contain the same values. As mentioned before, a single user may have multiple entries in this table, each corresponding to his identities with different Authentication Servers. However, all these entries must be linked to the user’s identity in the Shibboleth’s LDAP directory.

**FameShibLDAPServer**

This parameter specifies the name (or IP address) of the LDAP Server that Shibboleth uses for storing user attributes.

Default value: localhost.

**FameShibLDAPPort**

This parameter specifies the port the Shibboleth LDAP Server is running on.

Default value: 389.

**FameShibLDAPDN**

This mandatory parameter specifies the distinguished name of the user bound to the Shibboleth LDAP directory. The user must have the write privileges for the ‘loa’ attribute stored in the directory.

Example: ‘cn=Manager,dc=example,dc=com’.

Default value: none. The Fame module will attempt an anonymous binding if this item is not configured. It will almost certainly fail when an update to the ‘loa’ attribute is attempted later on, so it is strongly advised to set up this parameter.

**FameShibLDAPPassword**

This mandatory parameter specifies the password for the above DN.

Default value: none.

**FameShibLDAPBaseDN**

This mandatory parameter specifies the sub-tree of the Shibboleth LDAP directory where searches for the users should start from.

Example: ‘ou=shib-users,dc=example,dc=com’.

Default value: none.
5.2 Database Configuration

The FAME database containing FAME configuration data consists of three tables: (1) the Secrets table, (2) the Authentication Servers table, and (3) the FAME users table. A database user with read-only access to this database is also required by the Fame module. An exemplar database called fls and a database user flsuser (with password flspassword) can be created as follows.

```sql
# Example:
CREATE DATABASE fls;
GRANT SELECT on fls.*
TO flsuser IDENTIFIED BY 'flspassword';
```

The names used for the database, the Secrets, Authentication Servers and FAME users tables, database user and password can be respectively set up for the FAME module in httpd.conf configuration file using directives FameDB, FameSecretsTable, FameASTable, FameUsersTable, FameDBUser and FameDBPassword.

Secrets table

The Secrets table stores the secret keys used by the F-LS and the F-SSO for creation of cookies passed between them (namely, the sso and the request-url cookies). Each key has a version number associated with it, which is implemented as the timestamp when the key was inserted into the database. The key versioning allows periodical updates of the secret key without invalidating current valid cookies created with previous keys. This table is configured via parameter FameSecretsTable and is shown as follows.

```sql
# Create secrets table
CREATE TABLE secrets (  
  secret_key text NOT NULL,
  date_time datetime NOT NULL default '0000-00-00 00:00:00'
);
```

Authentication Servers table

The Authentication Servers table stores information about configured/supported Authentication Server(s). Each row includes the url at which the AS is running, the authentication type (exemplar settings are Username/password, Kerberos, Browser certificate, Smart-card certificate) and are shown on the main FAME page as an option for the user to choose, the LoA provided by the AS, the secret key shared between the AS and the F-LS (which is used for encryption and decryption of the auth-request and auth-reply tokens), and an optional timestamp. The last (i.e. timestamp) column in the table is optional, and it is not used by the FAME module. This table is configured via parameter FameASTable and is shown as follows.

```sql
# Create Authentication Servers table:
CREATE TABLE authservers (  
  url varchar(100) NOT NULL default '',
);```
FAME Users table

The Users table stores information about mappings between various identities the FAME users have with different Authentication Servers and their identity in the Shibboleth LDAP directory, where the user’s current LoA is stored and later picked up by Shibboleth. Each row contains the user’s LDAP id, the name of the LDAP attribute that this id refers to (such as `uid`, or `cn`) and the user’s alternative id with an Authentication Server set up by the IdP. This table is configured via parameter `FameUsersTable` and is shown as follows.

```sql
# Example:
CREATE TABLE fameusers (
  ldap_id varchar(255) NOT NULL default '',
  ldap_attribute varchar(100) NOT NULL default '',
  alternative_id varchar(255) NOT NULL default ''
);
```

5.3 Authentication Server(s) Configuration

The FAME system attempts to use existing authentication systems and to integrate with them with minimum modifications. We distinguish two cases of integration based on how the Authentication Server (AS) is implemented.

1. If AS is a custom-built system, then some modifications are required on the AS’s side, in order to accommodate requests passed by the F-LS component of the FAME system and return the necessary information back to the F-LS upon authentication. In this case, it is necessary to understand how information is passed between the F-LS and the AS in order to make the necessary modifications to the AS.

The AS receives the authentication request from the F-LS in the form of:

```
https://<address_of_the_AS>?AuthRequestToken=<encrypted_auth_request_token>
```

This means that the encrypted `auth-request` token is passed to the AS as an url parameter. It contains two parameters and is encrypted by the F-LS with the FLS_AS_KEY, a symmetric key shared between the F-LS and AS, before it is passed to the AS. The AS needs to decrypt the token and extract the two parameters contained in it: the random challenge (`RC`) and the return address of the F-LS (i.e. to where to redirect the user upon successful authentication), which are delimited by a comma (","). The format of the `auth-request` token is shown as follows:

```
<auth_request_token> = <random_challenge>,<address_of_the_FLS>
```
Upon successful authentication, the AS is required to redirect the user back to the F-LS and passes to it the auth-reply token via URL, which looks like the following:

https://<address_of_the_FLS>/?AuthReplyToken=<encrypted_auth_reply_token>

The auth-reply token is encrypted by the AS with the same shared symmetric key FLS_AS_KEY. It contains the AS’s response to the random challenge from the auth-request token (i.e. RC + 1) and the name of the authenticated user that has to be passed to Shibboleth (refer to Section 4.1 for information on what is considered as the user name), delimited by a colon (":"). The auth-reply token has the following format:

<auth_reply_token> = <random_challenge_response>:<user_name>
<encrypted_auth_reply_token> = E_{FLS,AS,KEY}(<auth_reply_token>)

Let us suppose you are running Kerberos Authentication Server using Apache module mod_auth_kerb. You should create a <Location> in your httpd.conf that is tied to our AS.pm script and protected by Kerberos. Such a section may look something like the following:

```plaintext
# Location protected by Kerberos
<Location /kerb-auth>
    SSLOptions +StrictRequire
    SSLRequireSSL
    PerlResponseHandler MyApache2::AS
    AuthType KerberosV5
    AuthName "Kerberos Login"
    KrbAuthRealms CS.MAN.AC.UK
    Krb5Keytab /etc/apache2/apache2_kerb.keytab
    KrbMethodK5Passwd on
    KrbServiceName HTTP
    KrbVerifyKDC on
    require valid-user
</Location>
```

The above code defines a <Location> within your Apache Web server served by the script AS.pm and accessible from a Web browser as https://<your_server>/kerb-auth only by a user successfully authenticated.
by Kerberos and using an SSL-protected connection. The <Location> /kerb-auth plays the actual role of your AS. The AS.pm script serving this <Location> will receive requests from the F-LS (but only if the user has been authenticated by the set method), perform the necessary tasks, and redirect the user back to the F-LS.

The AS.pm script can be reused for any authentication system other than Kerberos. The only thing that needs to be configured inside the script is the path to the file containing the Base64-encoded secret key shared between the F-LS and AS (look for variable $KEY_FH in the source code of AS.pm).

6 FAME and Shibboleth Integration

After configuring the FAME module using the instructions given in Section 5, the integration of the FAME module with the Shibboleth’s IdP should proceed according to the following steps:

(1) Extend the Shibboleth LDAP directory’s schema to include definitions of the two FAME-defined attributes: the ‘loa’ attribute and its expiration time;
(2) Insert the <SimpleAttributeDefinition> element in Shibboleth’s resolver.xml to define the ‘loa’ attribute and the corresponding <JNDIDirectoryDataConnector> element to tell Shibboleth how and from which source to pull the loa attribute.
(3) Modify the Shibboleth IdP’s ARP (Attribute Release Policy) located in file site.arp.xml to specify which requesting SPs should the loa attribute be released to.

6.1 Extending the Shibboleth LDAP Directory Schema

The inetOrgPerson LDAP object class is widely used in LDAP directories to represents people within an organisation and has been endorsed by Shibboleth to store users in its LDAP store. The FAME system has extended this object class by defining a new LDAP object class, called famePerson. The famePerson object class inherits all the attributes from the inetOrgPerson class and, in addition, has two newly defined attributes: nist-loa and nist-loaExpires. The first (new) attribute is used to store the current ‘loa’ value for the user, while the second stores the expiration time for the current ‘loa’ value (which is equal to the duration of the authenticated user’s SSO session). The prefix ‘nist’ in the attribute name is used to denote the NIST E-Authentication Guideline\(^\text{13}\) that our loa attribute definition conforms to. Shibboleth IdP is subsequently configured to pick up the nist-loa attribute from the LDAP store and pass it over to the requesting SP. The nist-loaExpires attribute is currently not passed to the SP, but is included in the LDAP store for potential future use.

The famePerson object class and the nist-loa and nist-loaExpires attributes are defined in the LDAP schema file, called fame.schema, which can be found in the FAME installation kit.

# FAME LDAP schema.
# The attribute types and object class in this schema include the
# specifications of the 'loa' and 'loaExpires' attributes and the
# specification of the 'famePerson' object class. The 'famePerson'
# object class is an extension of the general purpose 'inetOrgPerson'
# object class, and additionally contains the two newly defined
# attributes 'loa' and 'loaExpires'.

# LoA attribute definition.
attributeType ( 1.2.826.0.1.3344810.1.1.104 NAME 'nist-loa'
DESC 'The Level of Authentication Assurance conforming to the
NIST E-Authentication Guideline'
EQUALITY integerMatch
ORDERING integerOrderingMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.27
SINGLE-VALUE)

# LoA's expiration date attribute definition.
attributeType ( 1.2.826.0.1.3344810.1.1.106 NAME 'nist-loaExpires'
DESC 'Expiration date for the current LoA attribute'
EQUALITY caseExactMatch
ORDERING caseExactOrderingMatch
SYNTAX 1.3.6.1.4.1.1466.115.121.1.15
SINGLE-VALUE)

# famePerson object class definition.
objectClass ( 1.2.826.0.1.3344810.1.0.24 NAME 'famePerson'
DESC 'Person that uses FAME for authentication'
SUP inetOrgPerson
MAY ( nist-loa $ nist-loaExpires )
)

To add these new object and attribute definitions to the IdP’s LDAP directory, _fame.schema_ has to be copied into the _schema_ directory of the LDAP installation (on our system `/etc/openldap/schema`), where other LDAP schemas are stored as well. Then the copied schema has to be included in the LDAP’s configuration by inserting the line #4 below in the IdP LDAP Server’s configuration file (typically `/etc/openldap/slapd.conf`):

```plaintext
... include /etc/openldap/core.schema
include /etc/openldap/cosine.schema
include /etc/openldap/inetorgperson.schema
include /etc/openldap/fame.schema
...
```

### 6.2 Configuring Attribute Definitions and JNDI Data Connectors

A Shibboleth IdP sends all the attribute values to an SP through the use of a specialised _attribute resolver_ defined in the file _resolver.xml_. To send an attribute value to the SP, the value has to be converted to a SAML-based XML format and included in the _resolver.xml_ file in the form of the `<SimpleAttributeDefinition>` element. Next, a `<JNDIDirectoryDataConnector>` element has to be defined and referred to by the just created loa’s `<SimpleAttributeDefinition>` element, in order to tell Shibboleth how to pull the ‘loa’ attribute value from a data store (in our case an LDAP directory).
To define the `<SimpleAttributeDefinition>` element for the ‘loa’ attribute, insert the following in `resolver.xml`:

```xml
...<SimpleAttributeDefinition id="urn:oid:1.2.826.0.1.3344810.1.1.104" sourceName="nist-loa">
  <DataConnectorDependency requires="directory"/>
</SimpleAttributeDefinition>
...
```

The above code defines the attribute whose unique URN-like name is “urn:oid:1.2.826.0.1.3344810.1.1.104” (derived from the nist-loa attribute’s unique oid) and which uses a DataConnector with an id “directory” to obtain the attribute value. To define such a DataConnector, insert the following in `resolver.xml`:

```xml
...<JNDIDirectoryDataConnector id="directory">
  <Search filter="uid=%PRINCIPAL%">
    <Controls searchScope="SUBTREE_SCOPE" returningObjects="false"/>
  </Search>
  <Property name="java.naming.factory.initial" value="com.sun.jndi.ldap.LdapCtxFactory"/>
  <Property name="java.naming.provider.url" value="ldap://rpc56.cs.man.ac.uk/dc=rpc56,dc=cs,dc=man,dc=ac,dc=uk"/>
</JNDIDirectoryDataConnector>
...
```

The above DataConnector element has an id="directory", connects to an LDAP directory defined by the url "://rpc56.cs.man.ac.uk/" and the LDAP directory root "dc=rpc56,dc=cs,dc=man,dc=ac,dc=uk", and uses a search filter "uid=%PRINCIPAL%" to search for the user when searching for the %PRINCIPAL%’s (i.e. user’s) attributes. Modify the `<Property>` element to correspond to your LDAP server’s settings.

### 6.3 Configuring Shibboleth IdP Attribute Release Policy

The Shibboleth’s ARP (Attribute Release Policy) determines which of the defined attribute values finally gets released to which requesting SPs. It acts as a filter for the attribute values stored in the LDAP directory – ARPs can only be used to release the attribute values that are already stored in the LDAP directory and defined in `resolver.xml`; it can only be used to limit what information gets released to whom. On the other hand, the attributes must be defined in both `resolver.xml` and specified in the site’s ARP in order for the values to be passed to a requesting SP via Shibboleth.

The simplest configuration for the `loa` attribute is to define a site policy in `arp.site.xml` file. Policies stored in this file apply for the whole IdP’s site, i.e. for every user for whom this IdP retrieves/releases information. In order to configure a simple policy to release the `loa` attribute values to every requesting SP, the `arp.site.xml` should look like the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
```
<AttributeReleasePolicy
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="urn:mace:shibboleth:arp:1.0"
xsi:schemaLocation="urn:mace:shibboleth:arp:1.0 shibboleth-arp-1.0.xsd">
  <Description>Simplest possible ARP.</Description>
  <Rule>
    <Target>
      <AnyTarget/>
    </Target>
    <!-- Loa Attribute -->
    <Attribute name="urn:oid:1.2.826.0.1.3344810.1.1.104">
      <AnyValue release="permit"/>
    </Attribute>
  </Rule>
</AttributeReleasePolicy>