X/Open Preliminary Specification

X/Open Single Sign-On Service (XSSO) - Pluggable Authentication Modules

The Open Group

X/Open Preliminary Specification

X/Open Single Sign-On Service (XSSO) - Pluggable Authentication Modules

ISBN: X-XXXXX-XXX-X
Document Number: XXXX

Published in the U.K. by The Open Group, March 1997.

Any comments relating to the material contained in this document may be submitted to:

The Open Group
Apex Plaza
Forbury Road
Reading
Berkshire, RG1 1AX
United Kingdom

or by Electronic Mail to:

OGSpecs@opengroup.org

Contents

List of Figures

List of Tables

Preface

The Open Group

The Open Group is an international open systems organization that is leading the way in creating the infrastructure needed for the development of network-centric computing and the information superhighway. Formed in 1996 by the merger of the X/Open Company and the Open Software Foundation, The Open Group is supported by most of the world's largest user organizations, information systems vendors and software suppliers. By combining the strengths of open systems specifications and a proven branding scheme with collaborative technology development and advanced research, The Open Group is well positioned to assist user organizations, vendors and suppliers in the development and implementation of products supporting the adoption and proliferation of open systems.

With more than 300 member companies, The Open Group helps the IT industry to advance technologically while managing the change caused by innovation. It does this by:

- consolidating, prioritizing and communicating customer requirements to vendors
- conducting research and development with industry, academia and government agencies to deliver innovation and economy through projects associated with its Research Institute
- managing cost-effective development efforts that accelerate consistent multi-vendor deployment of technology in response to customer requirements
- adopting, integrating and publishing industry standard specifications that provide an essential set of blueprints for building open information systems and integrating new technology as it becomes available
- licensing and promoting the X/Open brand that designates vendor products which conform to X/Open Product Standards
- promoting the benefits of open systems to customers, vendors and the public.

The Open Group operates in all phases of the open systems technology lifecycle including innovation, market adoption, product development and proliferation. Presently, it focuses on seven strategic areas: open systems application platform development, architecture, distributed systems management, interoperability, distributed computing environment, security, and the information superhighway. The Open Group is also responsible for the management of the UNIX trade mark on behalf of the industry.

The X/Open Process

This description is used to cover the whole Process developed and evolved by X/Open. It includes the identification of requirements for open systems, development of CAE and Preliminary Specifications through an industry consensus review and adoption procedure (in parallel with formal standards work), and the development of tests and conformance criteria.

This leads to the preparation of a Product Standard which is the name used for the documentation that records the conformance requirements (and other information) to which a vendor may register a product. There are currently two forms of Product Standard, namely the Profile Definition and the Component Definition, although these will eventually be merged into one.

The X/Open brand logo is used by vendors to demonstrate that their products conform to the relevant Product Standard. By use of the X/Open brand they guarantee, through the X/Open Trade Mark Licence Agreement (TMLA), to maintain their products in conformance with the Product Standard so that the product works, will continue to work, and that any problems will be fixed by the vendor.

Open Group Publications

The Open Group publishes a wide range of technical literature, the main part of which is focused on specification development and product documentation, but which also includes Guides, Snapshots, Technical Studies, Branding and Testing documentation, industry surveys and business titles.

There are several types of specification:

- *CAE Specifications*

  CAE (Common Applications Environment) Specifications are the stable specifications that form the basis for our product standards, which are used to develop X/Open branded systems. These specifications are intended to be used widely within the industry for product development and procurement purposes.

  Anyone developing products that implement a CAE Specification can enjoy the benefits of a single, widely supported industry standard. In addition, they can demonstrate product compliance through the X/Open brand. CAE Specifications are published as soon as they are developed, so enabling vendors to proceed with development of conformant products without delay.

- *Preliminary Specifications*

  Preliminary Specifications usually address an emerging area of technology and consequently are not yet supported by multiple sources of stable conformant implementations. They are published for the purpose of validation through implementation of products. A Preliminary Specification is not a draft specification; rather, it is as stable as can be achieved, through applying The Open Group's rigorous development and review procedures.

  Preliminary Specifications are analogous to the *trial-use* standards issued by formal standards organizations, and developers are encouraged to develop products on the basis of them. However, experience through implementation work may result in significant (possibly upwardly incompatible) changes before its progression to becoming a CAE Specification. While the intent is to progress Preliminary Specifications to corresponding CAE Specifications, the ability to do so depends on consensus among Open Group members.

- *Consortium and Technology Specifications*

  The Open Group publishes specifications on behalf of industry consortia. For example, it publishes the NMF SPIRIT procurement specifications on behalf of the Network Management Forum. It also publishes Technology Specifications relating to OSF/1, DCE, OSF/Motif and CDE.

  Technology Specifications (formerly AES Specifications) are often candidates for consensus review, and may be adopted as CAE Specifications, in which case the relevant Technology Specification is superseded by a CAE Specification.

In addition, The Open Group publishes:

- *Product Documentation*

  This includes product documentation — programmer's guides, user manuals, and so on — relating to the Pre-structured Technology Projects (PSTs), such as DCE and CDE. It also includes the Single UNIX Documentation, designed for use as common product documentation for the whole industry.

- *Guides*

  These provide information that is useful in the evaluation, procurement, development or management of open systems, particularly those that relate to the CAE Specifications. The Open Group Guides are advisory, not normative, and should not be referenced for purposes of specifying or claiming conformance to a Product Standard.

- *Technical Studies*

  Technical Studies present results of analyses performed on subjects of interest in areas relevant to The Open Group's Technical Program. They are intended to communicate the findings to the outside world so as to stimulate discussion and activity in other bodies and the industry in general.

- *Snapshots*

  These provide a mechanism to disseminate information on its current direction and thinking, in advance of possible development of a Specification, Guide or Technical Study. The intention is to stimulate industry debate and prototyping, and solicit feedback. A Snapshot represents the interim results of a technical activity.

Versions and Issues of Specifications

As with all *live* documents, CAE Specifications require revision to align with new developments and associated international standards. To distinguish between revised specifications which are fully backwards compatible and those which are not:

- A new *Version* indicates there is no change to the definitive information contained in the previous publication of that title, but additions/extensions are included. As such, it *replaces* the previous publication.

- A new *Issue* indicates there is substantive change to the definitive information contained in the previous publication of that title, and there may also be additions/extensions. As such, both previous and new documents are maintained as current publications.

Corrigenda

Readers should note that Corrigenda may apply to any publication. Corrigenda information is published on the World-Wide Web at **http://www.opengroup.org/public/pubs**.

Ordering Information

Full catalogue and ordering information on all Open Group publications is available on the World-Wide Web at **http://www.opengroup.org/public/pubs**.

This Document

This document is a Preliminary Specification (see above).

- Chapter 1 is an introduction to the XSSO.
- Chapter 2 describes the conformance requirements.
- Chapter 3 describes the basic XSSO architecture.
- Chapter 4 describes the XSSO Sign-on services and the use of PAM and GSS-API to provide these.
- Chapter 5 describes the Parameter passing conventions and constants used within the PAM specification.
- Chapter 6 describes the application and system programming interfaces supported by PAM.
- Appendix A contains example header files.
- Appendix B discusses the topic of the administration of PAM Configuration.
- Appendix C discusses the issue of, internationalisation of the Sign-on Service.
- Appendix D describes the XSSO Account Management services.
- A glossary and index are provided.

Typographical Conventions

The following typographical conventions are used throughout this document:

- **Bold** font is used in text for filenames, and C-language keywords, type names, data structures and their members.
- *Italic* strings are used for emphasis or to identify the first instance of a word requiring definition. Italics in text also denote:
  — C-language variable names, for example, substitutable argument prototypes
  — C-language functions; these are shown as follows: *name*( ).
- Normal font is used for the names of constants and literals.
- The notation **<file.h>** indicates a header file.
- The notation [EABCD] is used to identify a C-language return code EABCD.
- Syntax, code examples and user input in interactive examples are shown in `fixed width` font.
- Variables within syntax statements are shown in *`italic fixed width font`*.
- Language-independent functions and arguments use ***bold italic*** font, for example, ***function***( ) and ***argument***.

Trade Marks

Kerberos™ is a trademark of the Massachusetts Institute of Technology.

OSF™ is a trademark of The Open Software Foundation, Inc.

X/Open® is a registered trademark, and the ''X'' device is a trademark, of X/Open Company Limited.

Acknowledgements

OpenGroup Security Program Group

Many members of the OpenGroup Security Program Group have contributed to this specification, either by providing additional material or by reviewing drafts.

Referenced Documents

The following documents are referenced in this specification:

RFC 1510
: Internet Proposed Standard, The Kerberos Network Authentication System, John Kohl, B.Clifford Neuman, issue 5.2, 21 April 1993.

CESG Memo
: CESG Memorandum No.1 Issue 1.2 Oct 1992, Glossary of Security Terminology.

Federal Criteria
: Federal Criteria Version 1.0 Dec 1992, Federal Criteria for Information Technology Security.

ISO/IEC 7498-2
: ISO/IEC 7498-2: 1989, Information Processing Systems — Open Systems Interconnection — Basic Reference Model — Part 2: Security Architecture.

ISO/IEC 10181
: ISO/IEC 10181, Information Technology — Open Systems Interconnection — Security Frameworks in Open Systems —

10181-1: Part 1: Security Frameworks Overview
10181-2: Part 2: Authentication Framework
10181-3: Part 3: Access Control
10181-4: Part 4: Non-repudiation Framework
10181-5: Part 5: Integrity Framework
10181-6: Part 6: Confidentiality Framework
10181-7: Part 7: Security Audit Framework

ITSEC
: Information Technology Security Evaluation Criteria, Provisional Harmonised Criteria, June 1991, Version 1.2, published by the Commission of the European Communities.

POSIX.0
: IEEE Std 1003.0/D15, June 1992, Draft Standard for Information Technology — Portable Operating System Interface (POSIX) — Part 0.

OSF RFC 86.0
: OSF RFC 86.0. October 1995, Unified Login with Pluggable Authentication Modules (PAM).

PAM Manual
: SUN, PAM Manual Pages.

The following X/Open documents are referenced in this specification:

Base GSS-API
: CAE Specification, December 1995, Generic Security Service API (GSS-API) Base (ISBN: 1-85912-131-4, C441).

XDSF
: Guide, December 1994, Distributed Security Framework (ISBN: 1-85912-071-7, G410).

The following background reading is referenced in this specification:

Lai and Samar
: Charlie Lai and Vipin Samar, "Making Login Services Independent of Authentication Technologies", ????, 1996

Chapter 1

Single Sign-On - Introduction

As IT systems proliferate to support business processes, users and system administrators are faced with an increasingly complicated interface to accomplish their job functions. Users typically have to sign-on to multiple systems, necessitating an equivalent number of sign-on dialogues, each of which may involve different usernames and authentication information. System administrators are faced with managing user accounts within each of the multiple systems to be accessed in a co-ordinated manner in order to maintain the integrity of security policy enforcement. This legacy approach to user sign-on to multiple systems is illustrated in Figure 1-1.

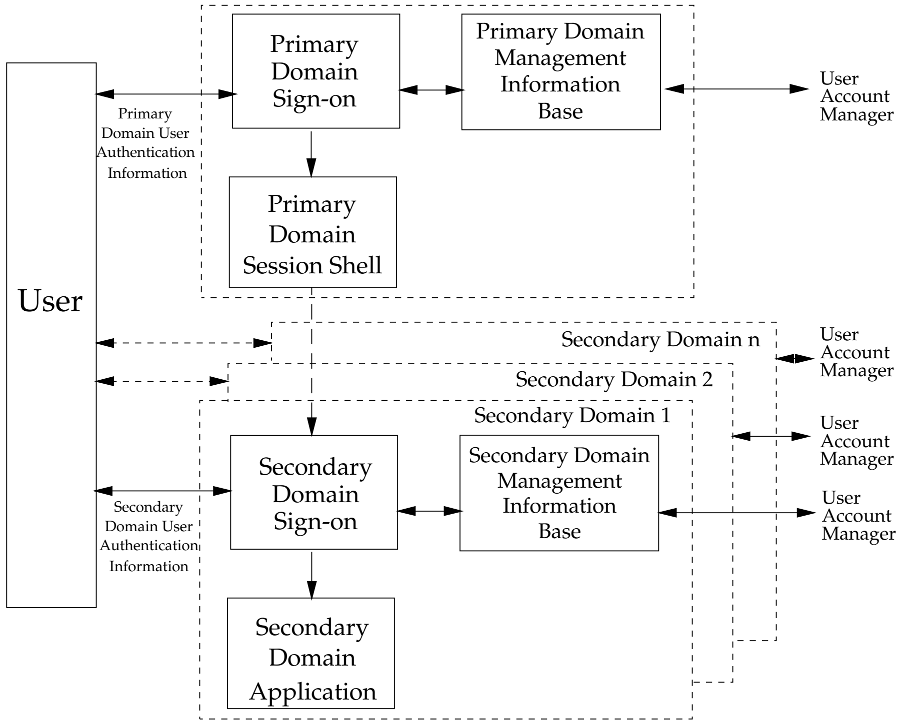


Figure 1-1 Legacy Approach to User Sign-On To Multiple Systems

Historically a distributed system has been assembled from components that act as independent security domains. These components comprise individual platforms with associated operating systems and applications.

These components act as independent domains in the sense that an end-user has to identify and authenticate himself independently to each of the domains with which he wishes to interact. This scenario is illustrated in Figure 1-1.

The end user interacts initially with a Primary Domain to establish a session with that primary domain. This is termed the *Primary Domain Sign-On* in Figure 1-1 and requires the end user to supply a set of user authentication information applicable to the primary domain, for example a username and password. The primary domain session is typically represented by an operating system session shell executed on the end user's workstation within an environment representative of the end user (e.g., process attributes, environment variables and home directory). From this primary domain session shell the user is able to invoke the services of the other domains, such as platforms or applications.

To invoke the services of a secondary domain an end user is required to perform a *Secondary Domain Sign-on*. This requires the end user to supply a further set of user authentication information applicable to that secondary domain. An end user has to conduct a separate sign-on dialogue with each secondary domain that the end user requires to use.

From the management perspective the legacy approach requires independent management of each domain and the use of multiple user account management interfaces.

Considerations of both usability and security give rise to a need to co-ordinate and where possible integrate user sign-on functions and user account management functions for the multitude of different domains now found within an enterprise. A service that provides such co-ordination and integration can provide real cost benefits to an enterprise through:

- reduction in the time taken by users in sign-on operations to individual domains, including reducing the possibility of such sign-on operations failing through user error.
- improved security through the reduced need for a user to handle and remember multiple sets of authentication information.
- reduction in the time taken, and improved response, by system administrators in adding and removing users to the system or modifying their characteristics.
- improved security through the enhanced ability of system administrators to maintain the integrity of user account configuration including the ability to inhibit or remove an individual user's access to all system resources in a co-ordinated and consistent manner.

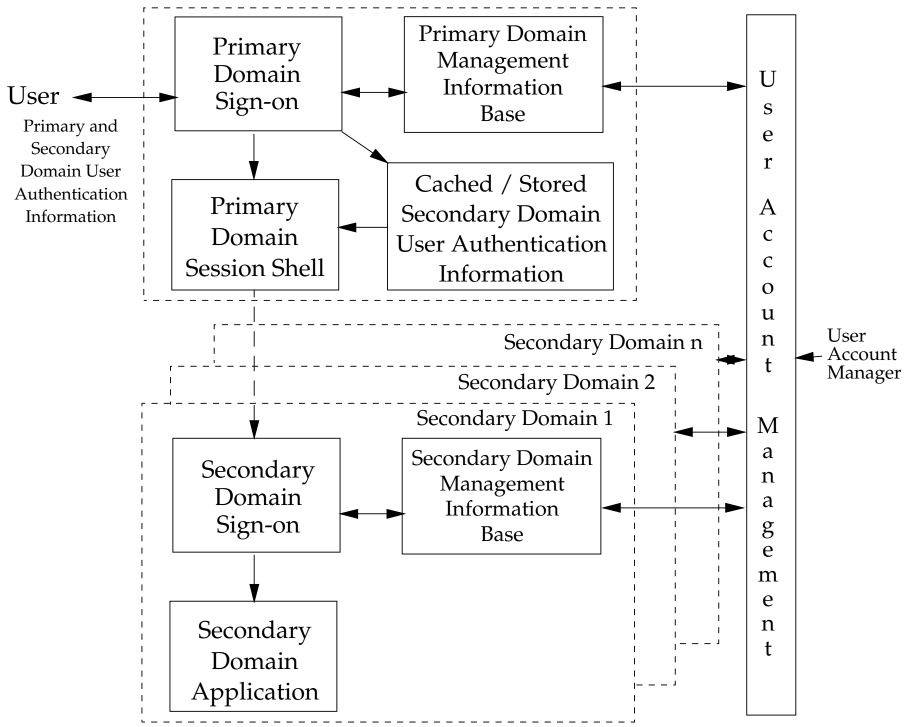


Figure 1-2 Single User Sign-On To Multiple Services

Such a service has been termed *Single Sign-On* after the end-user perception of the impact of this service. However, it is not intended to preclude prompting a user for additional information if it is required. An alternative appropriate term that could be used is *Integrated Sign-on*. In addition, both the end-user and management aspects of the service are equally important.

The Single Sign-On approach is illustrated in Figure 1-2. In the single sign-on approach the system may collect from the user as, part of the primary sign-on, all the identification and user authentication information necessary to support the authentication of the user to each of the secondary domains that the user may potentially require to interact with. The information supplied by the user is then used by *Single Sign-On Services* within the primary domain to support the authentication of the end user to each of the secondary domains with which the user actually requests to interact.

The information supplied by the end-user as part of the Primary Domain Sign-On procedure may be used in support of secondary domain sign-on in several ways:

- Directly, the information supplied by the user is passed to a secondary domain as part of a secondary sign-on.
- Indirectly, the information supplied by the user is used to retrieve other user identification and user authentication information stored within the a single sign-on management information base. The retrieved information is then used as the basis for a secondary domain sign-on operation.
- Immediately, to establish a session with a secondary domain as part of the initial session establishment. This implies that application clients are automatically invoked and communications established at the time of the primary sign-on operation.
- Temporarily stored or cached and used at the time a request for the secondary domain services is made by the end-user.

From a management perspective the single sign-on model provides a single user account management interface through which all the component domains may be managed in a co-ordinated and synchronised manner.

From an integration perspective significant security aspects of the Single Sign-On model are:

- the secondary domains have to trust the primary domain to:
  - correctly assert the identity and security attributes of the end user,
  - protect the authentication information used to verify the end user identity to the secondary domain from unauthorised use.
- The authentication information has to be protected when transferred between the primary and secondary domains against threats arising from interception or eavesdropping leading to possible masquerade attacks.

1.1 Scope of XSSO

The scope of the XSSO is application program interfaces in support of:

- the development of applications to provide a common, single end-user interface for an enterprise for sign-on, sign-off and password change operations, and
- the development of applications for the co-ordinated management of multiple user account management information bases maintained by an enterprise.

1.1.1 Functional Objectives

User Sign-on Interface

The following functional objectives have been defined for the XSSO in support of a user sign-on interface:

- The interface shall be independent of the type of authentication information handled and shall be capable of supporting all appropriate sign-on interactive sequences including none. For example, a smartcard may be used to supply a user identity to XSSO for the primary sign-on instead of prompting the user.
- Change of user controlled authentication information shall be supported. This is interpreted as initially being restricted to change of user password although capability for future extension shall not be precluded.
- Support shall be provided for an application calling the sign-on interface to establish a security context based upon a default user profile. User selection of a security context from a set of available user profiles is not required to be supported but shall not be precluded as a future extension.
- On session termination, or sign-off, the initiation of cleanup services that clean up at least the system on which the original sign-on was performed shall be supported.
- XSSO shall not require that all sign-on operations are performed at the same time as the primary sign-on operation. This otherwise would result in the creation of user sessions with all possible services even though those services may not actually be required by the user.
- Support shall be provided for the management of system sign-on policy.
- XSSO APIs must not preclude the use of 16 bit characters, although other system components may preclude their use in particular instances.
- The automatic refresh of credentials used for sign-on operations shall be supported.
- Both administratively controlled and algorithmic mappings between primary and secondary sign-on naming systems shall be supported.

Account Management Interface

The following functional objectives have been defined for the XSSO in support of an account management interface:

- The creation, deletion, and modification of accounts shall be supported.
- The setting of attributes for individual accounts shall be supported. The attributes to be supported shall include as a minimum those necessary to support the XBSS.

1.1.2 Non-Functional Objectives

The non-functional objectives of the XSSO are:

- The XSSO shall be authentication technology independent. The interface shall not prescribe the use of a specific authentication technology, nor preclude the use of any appropriate authentication technology.

  Note: Some authentication technologies, for example those based upon challenge-response mechanisms of which a user held device is a component may not be appropriate for use as part of *secondary sign-on* functions without invoking further user interaction.

- XSSO shall be independent of platform or operating system. XSSO shall not preclude the integration of common desktops or common servers, including mainframes. There is no expectation that such desktops or servers shall be capable of integration within XSSO without modification.
- It shall be possible to integrate legacy applications into the XSSO framework, and XSSO shall be designed to facilitate this. It may, however, be necessary for changes to be made to those applications at the level of the source code
- XSSO shall support sign-on to a client component of legacy client-server systems on both the local and remote platforms.

1.1.3 Security Objectives

The security objectives to be met by the XSSO are:

- XSSO shall not require the introduction of a single point of failure into a system.
- XSSO shall not adversely impact the availability of any individual system service.
- XSSO shall support the authorisation policies enforced by the constituent applications of the policy domain. That is, a principal shall only be able to obtain access via the services of XSSO to the same account information pertaining to an application that the principal is authorised to access using the services of the application itself.
- The XSSO shall not preclude the audit of all security relevant events which occur within the context of the XSSO.
- An XSSO implementation shall protect all security relevant information supplied to or generated by the XSSO implementation such that other services may adequately trust the integrity and origin of all security information provided to them as part of a secondary sign-on operation.
- An XSSO implementation shall provide protection to security relevant information when exchanged between its own constituent components and between those components and other services.

1.1.4 Out of Scope - End-user Sign-on Interface

The following aspects are not considered to be within the current scope of XSSO:

- Support for single sign-on across policy domain administrative boundaries.
- User initiated change of non-user configurable authentication information, for example magnetic badges, smartcards, etc.
- Password synchronisation between multiple authentication domains.
- Facilities for a user to claim a subset of permitted attributes when signing on or when initiating a secondary sign-on operation.
- Configuration and management of alternative sets of user profiles.
- Boot-time passwords. XSSO only applies to a system with a fully operational operating system.
- Facilities to support a change of user identity associated with an active session.
- Dependency of authentication method upon location or user identity or both. That is, the dynamic specification of the authentication mechanism.
- Multiple namespaces for primary sign-on. The user is not required, nor able, to select the primary domain for which the sign-on name is supplied. This is controlled by the XSSO sign-on configuration.
- Control of the sequence of secondary sign-on operations once the primary sign-on is completed. XSSO only controls the sequence of operations, including secondary sign-ons and credential acquisition, during the primary sign-on operation. Once this is completed the sequence of secondary sign-on operations is under the control of the end-user.
- Facilities for an end-user to configure services into XSSO. The configuration of XSSO is an administrative facility only.
- Maintenance of the consistency of the single sign-on account information base with underlying individual service account information bases when those underlying user account information bases are modified by means other than XSSO provided functionality. It is assumed that all account information bases are managed via the XSSO service.
- Graphical and command line user interfaces to XSSO based services. These are the province of applications written to utilise the XSSO.
- The definition of protocols to support interoperability between components from different XSSO implementations.

1.1.5 Out of Scope - Account Administration Interface

The specification of an interface and protocols to support the co-ordinated management of multiple account management information bases is deferred to a future specification.

These services are included in the description of the architecture Chapter 3 for contextual information. Also, Appendix D provides an introduction to the scope of the services to be covered by a future specification for Account Management Interfaces.

Chapter 2

Conformance Statement

2.1 XSSO (Base) Conformance

This section defines conformance criteria for implementations of the XSSO.

The following XSSO implementation conformance categories are defined:

- **PAM Application Programming Interface Conformance**
  This is applicable to an implementation of the PAM infrastructure that supports the PAM application programming interfaces that may be used by a sign-on application.
- **PAM System Programming Interface Conformance**
  This is applicable to an implementation of the PAM infrastructure that uses and supports the PAM system programming interfaces for the integration of PAM modules.
- **PAM Module Conformance**
  This is applicable to implementations of PAM modules to be used underneath the PAM infrastructure. A set of options equivalent to the module types are supported thus, Authentication, Account Management, Session Management, Password Management, Mapping.

2.2 PAM Application Programming Interface Conformance

An implementation of the PAM Infrastructure that conforms with this conformance category shall support the following interfaces:

pam_acct_mgmt	pam_authenticate
pam_authenticate_secondary	pam_chauthtok
pam_close_session	pam_end
pam_get_data	pam_getenv
pam_get_envlist	pam_get_item
pam_get_mapped_authtok	pam_get_mapped_username
pam_get_user	pam_open_session
pam_putenv	pam_setcred
pam_set_data	pam_set_item
pam_set_mapped_authtok	pam_set_mapped_username
pam_start	pam_strerror

2.3 PAM System Programming Interface Conformance

An implementation of the PAM Infrastructure that conforms with this conformance category shall support the following interfaces:

pam_sm_acct_mgmt	pam_sm_authenticate
pam_sm_authenticate_secondary	pam_sm_chauthtok
pam_sm_close_session	pam_sm_get_mapped_authtok
pam_sm_get_mapped_username	pam_sm_set_mapped_authtok
pam_sm_set_mapped_username	pam_sm_open_session
pam_sm_setcred	

2.4 PAM Module Conformance

An implementation of a PAM module that conforms with this conformance category shall support one or more of the following sets of interfaces:

Authentication Option
pam_sm_authenticate pam_sm_setcred

Account Management Option
pam_sm_acct_mgmt

Session Management Option
pam_sm_close_session pam_sm_open_session

Password Management Module
pam_sm_chauthtok

Mapping Option
pam_sm_get_mapped_authtok pam_sm_get_mapped_username
pam_sm_set_mapped_authtok pam_sm_set_mapped_username

An implementor of a module shall define how that module is configured, and in particular, define the options that may be included in a PAM configuration entry for the module.

An implementor of a module that is capable of using mapping shall document whether the module provides support for mapping usernames, or passwords, or both.

Chapter 3

XSSO Architecture

This chapter presents an overview of the architectural concepts of the XSSO services. The services and the architecture are described in greater detail in Chapter 4 and Appendix D on page 117. As described in Chapter 1 there are two perspectives to a Single Sign-On service, an end user perspective and a user account management perspective. For simplicity these two perspectives are presented as two separate models.

3.1 XSSO Single Sign-on Model

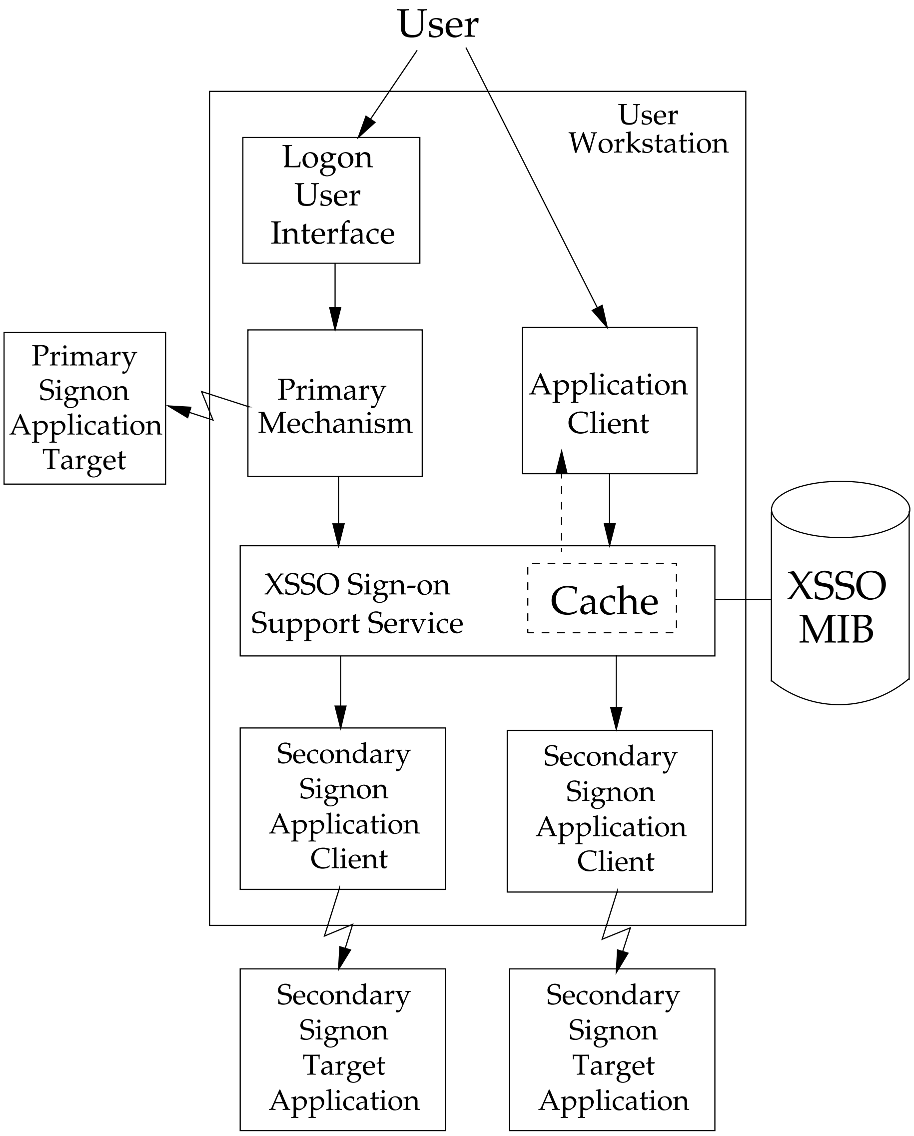


Figure 3-1 SSO Sign-On Model

Figure 3-1 is a top-level view of the sign-on model. The model illustrates a combination of primary and secondary sign-on operations.

The initial user sign-on is performed by the primary sign-on application. Secondary sign-on operations are invoked when a user invokes an application that interfaces to services that require user authentication. These services are typically client-server applications requiring the communication of user authentication information to a further platform.

Logon User Interface
The user interacts with the sign-on service via an interface provided by the application providing the system entry service that establishes a user session on the workstation. This application invokes the primary authentication mechanism, and any necessary secondary authentication mechanisms via the XSSO Services.

Primary Mechanism
The primary mechanism is used to authenticate the user as part of the system entry service. If this authentication fails then the user is generally denied access to the workstation. The primary mechanism may need to interact with a remote authentication service target (Primary Sign-on Application Target) in order to perform user authentication.

XSSO Sign-on Support Services
Authentication mechanism independence is provided by the invocation of common XSSO sign-on services by the primary and secondary sign-on applications. The XSSO sign-on services support multiple components for implementing user authentication and session establishment whilst maintaining a common interface for the calling application.

XSSO Service Cache
The XSSO service cache provides temporary storage for sign-on information obtained or derived as part of the primary sign-on operation from which it can be retrieved for use in subsequent secondary sign-on operations during the current user session. The cache is cleared on termination of a user session.

XSSO Management Information Base
The XSSO sign-on service depends upon a set of sign-on service management information. This comprises configuration information for the XSSO sign-on service itself, for example which authentication mechanisms to use, together with the user account information required by those authentication mechanisms and the other supporting services.

Secondary Sign-on Applications
In addition to the primary sign-on, that essentially supports access to a user session on the workstation and applications executed within it, secondary sign-on operations to authenticate the user and establish sessions with other management domains are generally necessary within a distributed environment. These are supported by the XSSO services in a manner that is generally transparent to the end-user on whose behalf the secondary sign-on operations are undertaken. These secondary sign-ons may occur at the time of the primary sign-on or later as an application is invoked.

The XSSO services invoked by the Primary Sign-on Application are responsible for:

- conducting the dialogue with the user. XSSO services need to acquire from the user all the information necessary to perform, or needed to derive the information necessary to perform, both the primary sign-on and any subsequent secondary sign-ons so that secondary sign-ons may be transparent to the end-user.
- authenticating the user. This may include authentication to multiple authentication services at this time to support later secondary sign-on operations as determined by administrative policy.

- authorising the creation of a user session,
- initialising the user session including establishing the session security context. The XSSO service shall not preclude auditing the creation of a user session,
- caching the information necessary to support subsequent secondary sign-ons and supporting the retrieval of the cached information. The information to be cached includes any security tokens obtained as part of the authentication operations and may also include the original authentication information supplied by the user if that is required for secondary sign-on operations. Cache management may include the refreshing of secondary sign-on credentials.
- supporting a session close down policy. This may include the invocation of configured services on user session termination including the forced termination of any residual processing initiated within the session. The XSSO service shall not preclude the auditing of session termination.

  Note: Deferred authentication, for example batch processing, whether directly invoked or scheduled, is considered to comprise a separate session and is not within the scope of the current XSSO specification. This may be subject to an extension within a subsequent version of the specification.

The XSSO services invoked by a secondary sign-on application client and target service in effect comprise a Distributed XSSO Sign-on Service. The XSSO sign-on service invoked by the target service performs essentially the same functions as the XSSO services invoked by the Primary Sign-on Application. However, the user dialogue is replaced by an exchange of information with the application client. The XSSO sign-on services invoked by the application client are responsible for retrieving the information required for the exchange with the target service from the XSSO service cache created by the primary sign-on operation or from the Sign-on Service Management Information.

In addition, the XSSO sign-on services invoked by the application client and target service are responsible for protecting the sign-on information exchanged.

3.2 XSSO Account Management Model

Figure 3-2 illustrates the XSSO Account Management Model. The objective of the XSSO Account Management Service is to support the development of management applications that are capable of managing a set of distributed account information bases whilst providing a common administrator user interface. This is to be achieved by defining an XSSO Account Management API to be supported by management modules specific to each of the individual account information bases. This will enable an XSSO Management Application developer to provide agent applications that will interface to management services provided by each domain that supports the XSSO ACM-API.

The definition of the Account Management API is deferred to a future specification.

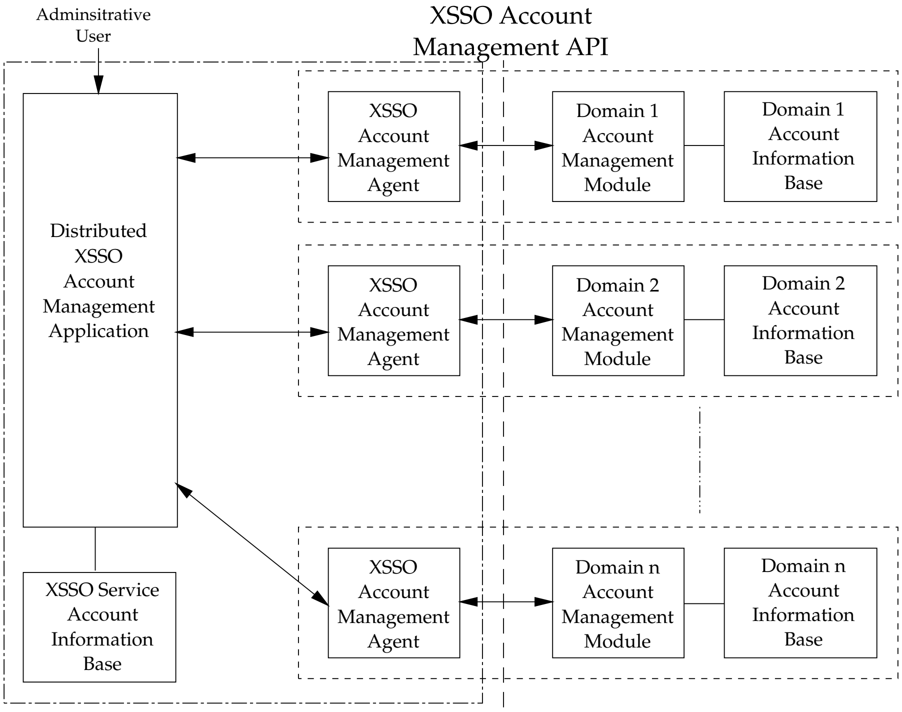


Figure 3-2 SSO Account Management Model

Chapter 4

XSSO Sign-on Services

4.1 XSSO Sign-on Service Structure

This specification defines an API to an XSSO Sign-on Service for use by sign-on applications. The XSSO Service API is independent of the specific authentication mechanisms used. There are two distinct aspects to a Single Sign-on service; the Primary Sign-on operation in which a user signs onto the policy domain as a whole, and Secondary Sign-on operations in which a user signs onto a service within the domain. The principal objective of a Single Sign-on service is that Secondary Sign-on operations may be transparent to the user.

The XSSO Service API comprises the PAM (Pluggable Authentication Modules) Service. The next subsection describes the PAM service.

The chapter concludes by providing illustrations of how the PAM service is used to support both Primary and Secondary Sign-on operations in support of the concept of Single Sign-on. There are four models of secondary sign-on considered:

- Single Sign-on to a secondary domain collocated with primary sign-on session, that is on the same host platform.
- Single Sign-on to a distributed authentication service to access a target application on a remote platform. This model uses the GSS-API or any such appropriate mechanism for authentication between the client and target applications. The secondary authentication uses a secure authentication protocol, for example the DCE security service or a GSS-API implementation over Kerberos.
- Single Sign-on to a local service on a remote platform. This involves a distributed application on the remote system acting as a proxy for the user principal and conducting the sign-on to the secondary local service.
- Single Sign-on to a secondary distributed authentication service. This involves a distributed application acting as proxy for the user principal and conducting the sign-on to the secondary distributed authentication service.

4.2 PAM Service Overview

This section provides an overview of the PAM service.

PAM (Pluggable Authentication Modules) provides system administrators with the flexibility to:

- choose any authentication service available on a system to perform end-user authentication for an application,
- use multiple authentication services thus providing a means of integrating authentication technologies with system-entry services,
- add new authentication service modules to a system and make them available without having to modify any applications.
- incorporate mapping services to map user names and authentication tokens between different authentication domains.

The PAM framework comprises an interface library and multiple PAM service modules. The PAM interface library is the layer implementing the Application Programming Interface (PAM-API). The PAM service modules are a set of dynamically loadable objects invoked by the PAM-SPI to provide a particular type of user authentication.

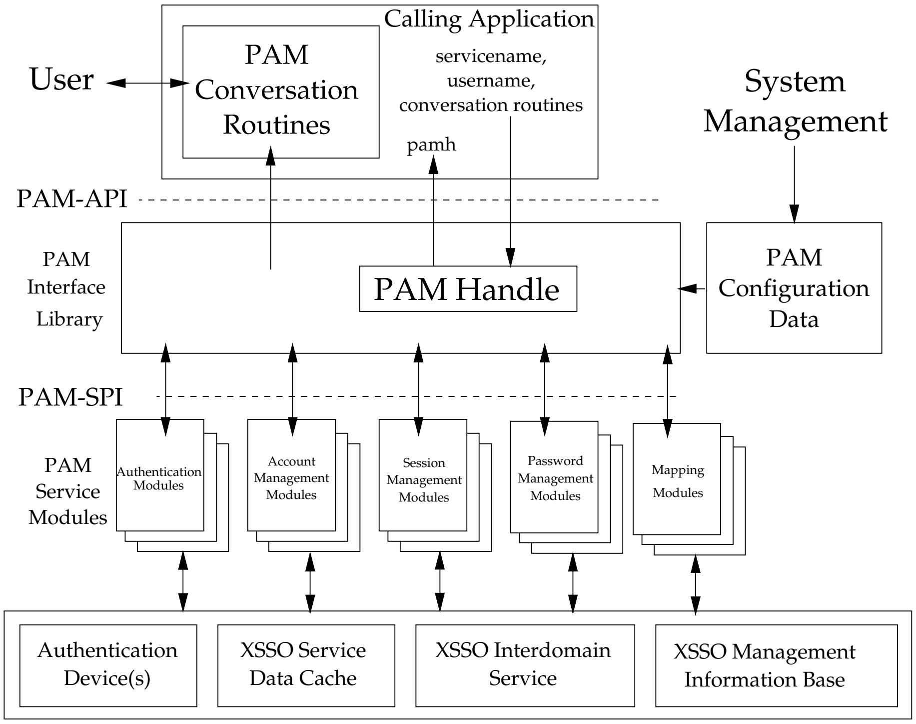


Figure 4-1 PAM Framework

Figure 4-1 illustrates the PAM framework. PAM supports five different types of service modules:

- Authentication
- Account Management
- Session Management
- Password Management
- Mapping

PAM may be configured with multiple instances of each module type to support sign-on operations to different authentication domains and the use of multiple mechanisms.

A system manager configures the PAM service via the PAM Configuration Data. This specifies which PAM service modules are to be called for a particular application. It also specifies the sequence, if more than one module is to be called, and the interdependencies of the results of each of the authentication operations.

A calling application is responsible for implementing the routines for conducting a dialogue with an end-user. This makes PAM independent of the user interface. The address of these callback communication functions, together with the servicename and the username are passed to the first PAM call. This call returns a handle to an internal PAM structure which maintains information related to the caller and via which information is passed to and returned by each of the PAM service modules that is invoked.

The user prompts used for a dialogue with an end-user are provided by the backend module and can therefore easily prompt for service specific information such as a PIN.

4.2.1 PAM-API

The functions comprising the PAM-API may be grouped into six categories of functions:

PAM Framework Layer Functions

These functions enable an application to invoke PAM service modules and to communicate information to the PAM Service Modules.

pam_start( )
pam_end( )
pam_get_data( )
pam_set_data( )
pam_get_item( )
pam_set_item( )
pam_getenv( )
pam_getenvlist( )
pam_putenv( )
pam_strerror( )

pam_start( ) and *pam_end*( ) are PAM transaction routines for establishing and terminating a PAM session. *pam_start*( ) takes as arguments the name of the application calling PAM, the name of the user to be authenticated and the address of the callback conversation structure provided by the caller. It returns a handle for use with subsequent calls to the PAM library.

pam_get_data( ) and *pam_set_data*( ) are routines for accessing and updating module specific data from the PAM handle.

pam_get_item( ) and *pam_set_item*( ) are routines for PAM that allow both applications and PAM service modules to access and update common PAM information such as service name, user name, remote host name, remote user name, etc. from the PAM handle. The values that may be manipulated by these functions are listed in Table 4-1.

Item Name	Description
PAM_SERVICE	Service name
PAM_USER	User name
PAM_RUSER	Remote user name
PAM_TTY	tty name
PAM_RHOST	remote host name
PAM_CONV	pam_conv structure
PAM_AUTHTOK	Authentication token
PAM_OLDAUTHTOK	Old authentication token.

Table 4-1 PAM Item Names

Note: the values of PAM_AUTHTOK and PAM_OLDAUTHTOK are only available to PAM modules and not to applications.

pam_getenv( ), *pam_getenvlist*( ) and *pam_putenv*( ) enable the calling application and PAM modules to set and retrieve environment variables for the user session which will be established with *pam_open_session*( ).

pam_strerror( ) returns error status information.

Authentication Functions

pam_authenticate( )
pam_authenticate_secondary( )
pam_setcred( )

The *pam_authenticate*( ) function is called to verify the identity of the current user. The *pam_authenticate_secondary*( ) function is called to authenticate a username in a secondary domain independently of the primary user authentication and user session establishment. The caller will typically have previously retrieved the username and authentication token to be used with the secondary target domain by calls to the mapping module.

pam_setcred( ) function is called to set the credentials of the current process associated with the authentication handle supplied. Typically, this is done after the user has been authenticated.

Account Management Functions

pam_acct_mgmt( )

This function is used to verify the authorisation of the user to sign-on. It will typically include checking for password and account expiration, valid login times, etc.

Session Management Functions

pam_open_session( )
pam_close_session( )

These functions are called on the initiation and termination of a PAM session. They may support session auditing.

Password Management Functions

pam_chauthtok( )

This function is called to change the authentication token (password) associated with the user.

Mapping Functions

pam_get_mapped_username( )
pam_get_mapped_authtok( )
pam_set_mapped_username( )
pam_set_mapped_authtok( )

These functions are called to set and retrieve identities and authentication tokens (e.g., passwords) that are associated with (mapped to) the specified identity.

4.2.2 PAM-SPI

The functions comprising the PAM-SPI are provided by the modules called by the PAM infrastructure and are grouped below on the basis of module type.

Authentication Module Functions

pam_sm_authenticate( )
pam_sm_authenticate_secondary( )
pam_sm_setcred( )

The *pam_sm_authenticate*( ) function is called to verify the identity of the current user. The *pam_sm_authenticate_secondary*( ) function is called to authenticate a username in a secondary domain independently of the primary user authentication and user session establishment. The caller will typically have previously retrieved the username and authentication token to be used with the secondary target domain by calls to the mapping module.

pam_sm_setcred( ) function is called to set the credentials of the current process associated with the authentication handle supplied. Typically, this is done after the user has been authenticated.

Account Management Module Functions

pam_sm_acct_mgmt( )

This function is used to verify the authorisation of the user to sign-on. It will typically include checking for password and account expiration, valid login times, etc.

Session Management Module Functions

pam_sm_open_session( )
pam_sm_close_session( )

These functions are called on the initiation and termination of a PAM session. They may support session auditing.

Password Management Module Functions

pam_sm_chauthtok( )

This function is called to change the authentication token (password) associated with the user.

Mapping Module Functions

pam_sm_get_mapped_username( )
pam_sm_get_mapped_authtok( )
pam_sm_set_mapped_username( )
pam_sm_set_mapped_authtok( )

These functions are called to set and retrieve identities and authentication tokens (e.g., passwords) that are associated with (mapped to) the specified identity.

4.2.3 PAM Configuration

PAM is controlled by a set of configuration information. An example set of configuration information is given in Table 4-2.

Service	Module_type	Control_flag	Module_path	Options
login	mapping	sufficient	libmap.so	
login	mapping	sufficient	libmapfoo.so	
login	auth	required	pam_unix_auth.so	nowarn
login	session	required	pam_unix_session.so	
login	account	required	pam_unix_account.so	
login	password	required	pam_unix_passwd.so	
ftp	auth	required	pam_skey_auth.so	debug
ftp	session	required	pam_unix_session.so	
telnet	session	required	pam_unix_session.so	
passwd	mapping	required	libmap.so	
passwd	mapping	required	libmapfoo.so	
passwd	password	required	pam_unix_passwd.so	
OTHER	auth	required	pam_unix_auth.so	
OTHER	session	required	pam_unix_session.so	
OTHER	account	required	pam_unix_account.so	
OTHER	password	required	pam_unix_passwd.so	

Table 4-2 PAM Configuration with Different Modules

The PAM configuration data is logically grouped into records comprising the following five fields:

Service
Service denotes the systems-entry application, for example login, passwd, rlogin. The name OTHER denotes the module used by default if a specific entry for a service is not specified.

Module_type
Module_type denotes the type of PAM module. Valid module types are *mapping*, *auth*, *account*, *session* and *password*.

Control_flag
The *control_flag* determines the behaviour of stacking multiple modules by specifying whether any particular module is *required*, *sufficient*, *requisite* or *optional*.

Module_path
The *Module_path* specifies the location of the module to be loaded.

Options
Options is used to define module-specific options that are passed to PAM modules.

Note: An general option to instruct a module to use a Personal Security Device (e.g., a smartcard) is not included in this specification. Such an option is considered to be unnecessary and an administrative complication. It is expected that specific PSD aware modules will be developed for any appropriate service and will be specified within the PAM configuration.

Service	Module_type	Control_flag	Module_path	Options
login	auth	required	pam_unix_auth.so	nowarn
login	auth	optional	pam_kerberos.so	
rlogin	auth	sufficient	pam_rhosts_auth.so	
rlogin	auth	required	pam_unix_passwd.so	

Table 4-3 PAM Configuration File with Stacked Modules

Table 4-3 illustrates the stacking of authentication modules within a PAM configuration file. The entries for *login* illustrate support for integrated authentication to both UNIX and Kerberos domains. The entries for *rlogin* illustrate support for alternative methods of authentication; rhosts based authentication is sufficient for rlogin but if that is not successful then normal UNIX authentication is required.

4.3 Models of Primary and Secondary Sign-on

4.3.1 Primary Sign-on

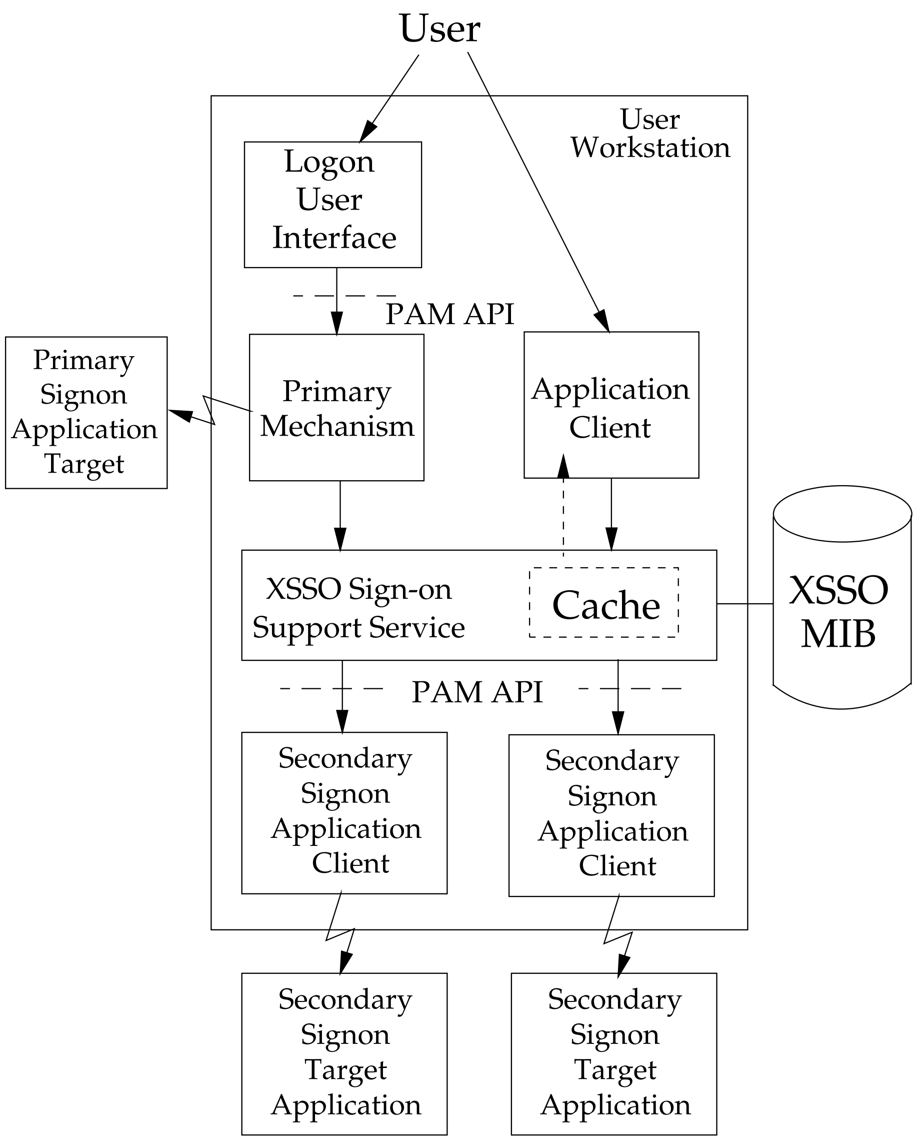


Figure 4-2 SSO Primary Sign-On

Figure 4-2 illustrates the basic model of the SSO Service in support of a Primary Sign-on operation. The essential features are:

- the primary domain sign-on application is responsible for invoking the PAM service via the PAM-API to perform user authentication. The primary domain sign-on application is also responsible for providing PAM support functions for prompting the user for sign-on information and for sending messages and error reports to the user. This makes XSSO independent of the user interface.

- the PAM Service is controlled by configuration data that define for a particular application which PAM modules are to be invoked by the PAM service for the particular Primary Domain Sign-On Application. Separate modules may be specified for authentication, authorisation, session control, mapping and authentication information (password) change. Multiple modules of each type may be configured for use by a particular application. That is, a single authentication dialogue may perform both local and distributed service authentication.
- the PAM modules may require to utilise the services of an authentication device, for example a badge or smartcard reader, or to retrieve additional information from a management information base, for example, for mapping usernames and passwords.
- the mechanism specific modules may cache information retrieved or derived during the primary sign-on operation, such as authentication credentials, in a manner that makes them accessible for subsequent use in secondary authentication operations by client applications executed within the same user session.
- in support of authentication to multiple domains as part of the primary sign-on, PAM modules may require to map information supplied by the user, or retrieved from the XSSO Management Information Base into representations applicable to the specific authentication domains. An example is the mapping of a user identity. This mapping service is supported by PAM mapping modules.
- PAM modules may require to retrieve information from the XSSO Account Management Information Base, for example authentication information, environment information etc. The retrieval of information from the XSSO Account Management information Base is subject to authorisation. The authorisation required may be dependent upon the information requested for retrieval.
- The PAM password modules may need to modify the authentication information held for an account within the XSSO Management Information Base

Note: The interfaces used by the PAM modules to manipulate the cache or access the XSSO Account Management Information Base are not included within the scope of this particular specification. They will be addressed as part of the Account Management Service in a future specification.

4.3.2 Secondary Sign-on

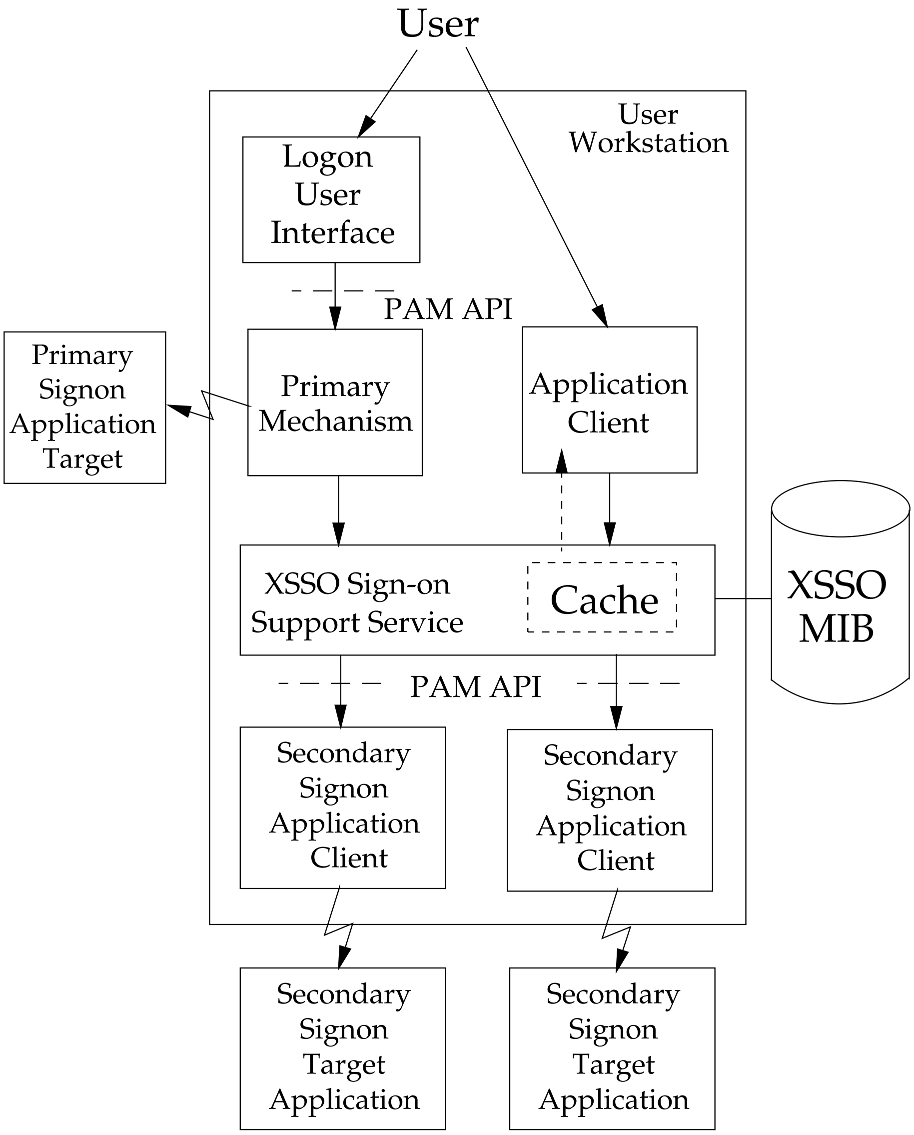
Four scenarios of secondary sign-on were identified at the beginning of this chapter. Each of these is described separately below.

Single Sign-on to Local Application Domain

Figure 4-3 Single Sign-On to Local Application Domain

Figure 4-3 illustrates the basic structure of a secondary sign-on operation to a local application domain. That is, sign-on to an application that resides on the same host as the primary sign-on operation.

In this case, the client application performs the secondary sign-on operation by retrieving information from the XSSO Service Data Cache that was placed there by the primary sign-on operation. This information may be used directly for sign-on purposes or may be used to retrieve additional information from the XSSO Management Information Base. These operations may require the mapping of the information into a representation applicable to the local application.



The secondary sign-on does not necessarily preclude any interaction with the user. Although the concept of Single Sign-on emphasises the transparency to users of the secondary sign-on operation, it does not preclude a security policy for a secondary sign-on requiring some interaction with the user.

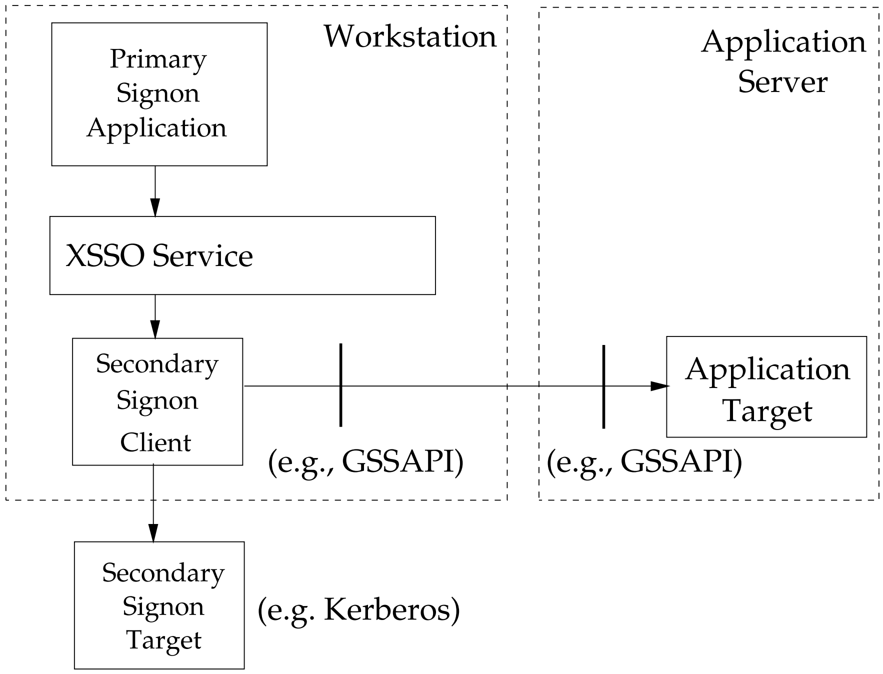


Figure 4-4 Single Sign-On to Distributed Domain

Single Sign-on to Distributed Domain

Figure 4-4 illustrates the basic structure of a secondary sign-on operation to an application based upon a distributed security service domain. That is, sign-on to an application that uses the services of a distributed authentication service for the purposes of association security context establishment between client and server components via the use of the GSS-API or any such appropriate authentication mechanism.

The client retrieves distributed authentication service credentials from the XSSO Service Data Cache and exchanges these with the server application using the GSS-API or any such appropriate mechanism. The server application is then responsible for establishing the appropriate processing environment for the client principal.

Single Sign-on to Remote Local Service

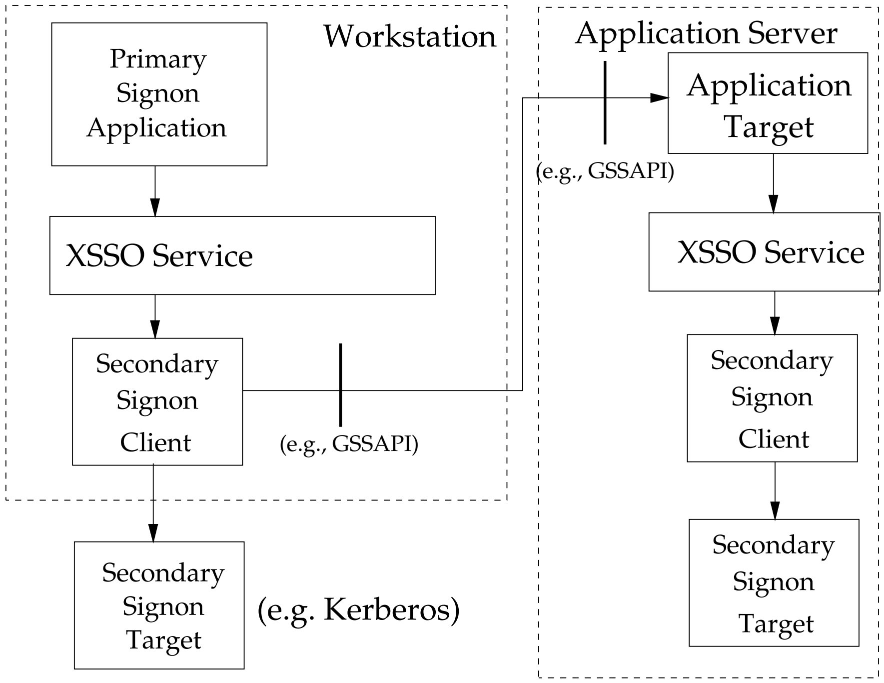


Figure 4-5 Single Sign-On to Remote Local Service

Figure 4-5 illustrates the use of PAM to support single sign-on to remote services that are not based on Distributed Security Services. That is, the client and server components are not part of the same authentication domain. Such services are often referred to as "*Legacy*" services and utilise an authentication service that is localised on the remote platform. An example is a traditional RDBMS implementation.

In this case an application on the target application server platform acts as proxy for the user principal. On the target platform PAM is utilised as if it was a Primary sign-on with the proxy application acting as sign-on principal. The proxy application uses PAM to retrieve the necessary authentication information from the XSSO Service Data Cache or the XSSO Management Information Base, or both. The proxy application may utilise the XSSO Interdomain Mapping service. As in the local application sign-on, additional interaction with the user or an authentication device may be required. This would need to be supported by the distributed application that is acting as proxy for the user.

Single Sign-on to Remote Distributed Service

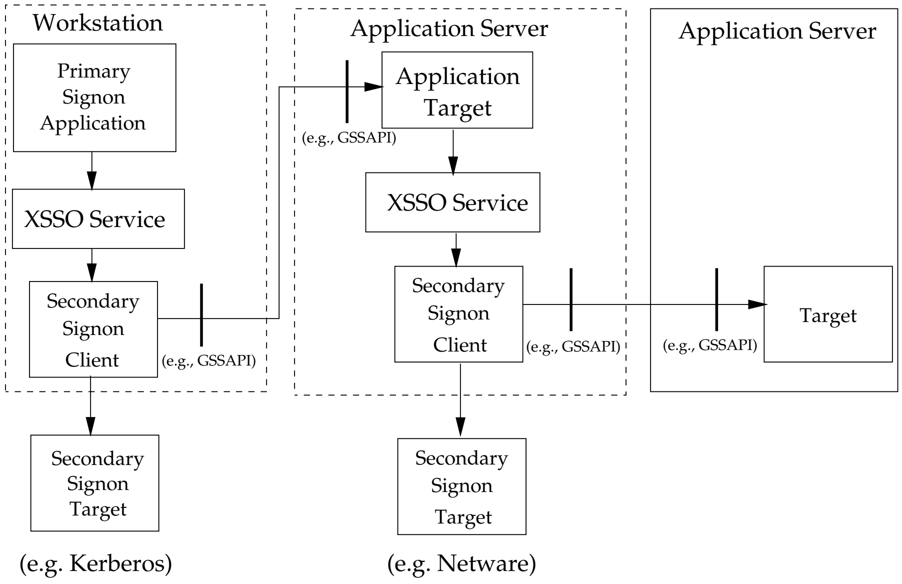


Figure 4-6 Single Sign-On to Remote Distributed Service

Figure 4-6 illustrates the use of PAM to support single sign-on to remote services that are based on a secondary Distributed Security Service. An example is a network operating system such as Netware.

In this case a client application acts as proxy for the user principal. On the target PAM is utilised as if it was a Primary sign-on with the proxy application acting as sign-on principal. The proxy client uses PAM to retrieve the necessary authentication information from the XSSO Service Data Cache or the XSSO Management Information Base, or both and may utilise the XSSO Interdomain Mapping service.

Chapter 5

Parameter Passing Conventions in PAM

This chapter describes the data types and constants used by the PAM functions. It also explains calling conventions for these functions.

5.1 Structured Data Types

Wherever these PAM-API C-bindings describe structured data, only fields that must be provided by all PAM-API implementations are documented. Individual implementations may provide additional fields, either for internal use within PAM-API routines, or for use by non-portable applications.

5.1.1 Messages

The structure **pam_message** is used to pass prompt, error message, or any text information from the PAM services to the application or user. It is the responsibility of the PAM service modules to localise the messages. The memory used by *pam_message* has to be allocated and freed by the PAM modules. The *pam_message* structure has the following structure:

```
struct pam_message{
       int    msg_style;
       char   *msg;        /* message */
};
```

The *msg_style* can be set to a number of values. See Table 5-2 on page 31. The structure **pam_response** is used to get the response back from the application or user. The storage used by *pam_response* has to be allocated by the application and freed by the PAM modules. It is defined as:

```
struct pam_response{
       char   *response;
       int    resp_retcode;
};
```

5.1.2 Call Back Information

The structure **pam_conv** contains the address of the conversation function provided by the application. The underlying PAM service module invokes this function to output information to and retrieve input from the user. The *pam_conv* structure has the following entries:

```
struct pam_conv{
   int (*conv) (int, struct pam_message **,
        struct pam_response **, void *);
   void *appdata_ptr;
};

   where
   int conv(int num_msg,
            const struct pam_message **msg, struct pam_response **resp,
            void *appdata_ptr);
```

The function, *conv*() is called by a service module to hold a PAM conversation with the application or user. For window applications, the application can create a new pop-up window to be used by the interaction.

The parameter, *num_msg* is the number of messages associated with the call. The parameter, *msg*, is a pointer to an array of length *num_msg* of the *pam_message* structure.

appdata_ptr is an application data pointer which is passed by the application to the PAM service modules. Since the PAM modules pass it back through the conversation function, the applications can use this pointer to point to any application-specific data.

5.1.3 Opaque Data Types

pam_handle
: This is opaque to the caller and returned to the caller upon initiation of a PAM session. It is subsequently passed as a parameter to each PAM-API call.

5.2 Status Values

One or more status codes are returned by each PAM-API routine. An implementation of PAM functions shall return PAM_SUCCESS and other status values appropriate for the implementation of the function. The characteristics of a particular implementation may make some status returns inappropriate for that implementation.

5.2.1 PAM Status Codes

PAM-API routines return PAM status codes as their **int** function value. These codes indicate major status errors that are independent of the underlying mechanism used to provide the security service.

Table 5-1 Routine Errors

Name	Value in Field	Meaning
[PAM_SUCCESS]	0	Successful completion.
[PAM_OPEN_ERR]	1	Failure when dynamically loading a service module.
[PAM_SYMBOL_ERR]	2	Symbol not found in service module.
[PAM_SERVICE_ERR]	3	Error in underlying service module.
[PAM_SYSTEM_ERR]	4	System error.
[PAM_BUF_ERR]	5	Memory buffer error.
[PAM_CONV_ERR	6	Conversation failure.
[PAM_PERM_DENIED]	7	The caller does not possess the required authority.
[PAM_MAXTRIES]	8	Maximum number of tries exceeded.
[PAM_AUTH_ERR]	9	Authentication error.
[PAM_NEW_AUTHTOK_REQD]	10	New authentication token required from user
[PAM_CRED_INSUFFICIENT]	11	Cannot access authentication database because credentials supplied are insufficient.
[PAM_AUTHINFO_UNAVAIL]	12	Cannot retrieve authentication information.
[PAM_USER_UNKNOWN]	13	The user is not known to the underlying account management module.
[PAM_CRED_UNAVAIL]	14	Cannot retrieve user credentials.
[PAM_CRED_EXPIRED]	15	User credentials have expired.
[PAM_CRED_ERR]	16	Failure setting user credentials.
[PAM_ACCT_EXPIRED]	17	User account has expired.
[PAM_AUTHTOK_EXPIRED]	18	Password expired and no longer usable.
[PAM_SESSION_ERR]	19	Cannot initiate / terminate a PAM session.
[PAM_AUTHTOK_ERR]	20	Error in manipulating authentication token.
[PAM_AUTHTOK_RECOVERY_ERR]	21	Old authentication token cannot be recovered.
[PAM_AUTHTOK_LOCK_BUSY]	22	The authentication token lock is busy.
[PAM_AUTHTOK_DISABLE_AGING]	23	Authentication token ageing is disabled.
[PAM_NO_MODULE_DATA]	24	Module data not found.
[PAM_IGNORE]	25	Ignore this module.
[PAM_ABORT]	26	General PAM failure.
[PAM_TRY_AGAIN]	27	Unable to complete operation. Try again.
[PAM_MODULE_UNKNOWN]	28	Module type unknown.
[PAM_DOMAIN_UNKNOWN]	29	Domain unknown.

5.3 Constants

The tables below set out the constants defined by the specification, and the value to which they are set.

Name	Value	Meaning
[PAM_PROMPT_ECHO_OFF]	1	Echo off when getting response.
[PAM_PROMPT_ECHO_ON]	2	Echo on when getting response.
[PAM_ERROR_MSG]	3	Error message.
[PAM_TEXT_INFO]	4	Textual information.
[PAM_MAX_NUM_MSG]	32	Maximum number of messages passed to the application through the conversation function call.
[PAM_MAX_MSG_SIZE]	512	Maximum size in characters of messages passed to application through the conversation function call.
[PAM_MAX_RESP_SIZE]	512	Maximum size in characters of each response passed from the application through the conversation function call.

Table 5-2 Message Constants

5.4 Flags

The tables below set out the flags defined by the specification, and the value to which they are set.

Name	Value	Meaning
General flags		
PAM_SILENT	0x80000000	Switch off messages from service
Flags for pam_authenticate		
PAM_DISALLOW_NULL_AUTHTOK	0x1	Disallow a NULL authentication token.
Flags for pam_setcred		
PAM_ESTABLISH_CRED	0x1	set user credentials for the authentication service.
PAM_DELETE_CRED	0x2	Delete user credentials from the authentication service.
PAM_REINITIALISE_CRED	0x4	Reinitialise user credentials.
PAM_REFRESH_CRED	0x8	Extend lifetime of user credentials.
Flags for pam_sm_chauthtok		
PAM_CRED_PRELIM_CHECK	0x1	Preliminary check for update readiness.
PAM_UPDATE_AUTHTOK	0x2	Update authentication token.
Flags for pam_sm_chauthtok and pam_chauthtok		
PAM_CHANGE_EXPIRED_AUTHTOK	0x4	Force a change to an expired authentication token.

Table 5-3 Flags

5.5 Item_type

The tables below set out the item_types defined by the specification, and the value to which they are set.

Name	Value	Meaning
PAM_SERVICE	1	The program service name.
PAM_USER	2	The user name.
PAM_TTY	3	The tty name.
PAM_RHOST	4	The remote host name.
PAM_CONV	5	The conversation structure.
PAM_AUTHTOK	6	The authentication token.
PAM_OLDAUTHTOK	7	The old authentication token.
PAM_RUSER	8	The remote user name.
PAM_USER_PROMPT	9	The user prompt.

Table 5-4 Item

5.6 PAM Configuration Entry Constants

Each entry has the following format:

```
<service_name> <module_type> <control_flag> <module_path> <options>
```

An entry commencing with a "#" character will be ignored.

The following subsections define the string constants that are used within a PAM configuration entry.

5.6.1 Service Name

The *service_name* denotes the service (for example, **login** , **dtlogin** , or **rlogin**). The service name is defined by the supplier of the application that is calling PAM.

The keyword, **other** , indicates the module that should be used for all applications which have not been included in the PAM configuration under a specific service name. The *other* keyword can also be used if all services using the same *module_type* have the same requirements.

5.6.2 Module Type

The PAM framework supports five module types. These are listed in Table 5-5.

Module_type	Module
auth	Authentication module
account	Account management module
mapping	Mapping module
password	Password manamgement module
session	Session management module

Table 5-5 Module Type

5.6.3 Control Flags

Control Flags
required
sufficient
requisite
optional

Table 5-6 Control Flags

The PAM framework processes each service module in the stack.
If a *requisite* module fails the PAM framework immediately returns to the application with the error returned by the *requisite* module and stops processing the module stack.

If all *requisite* and *required* modules in the stack succeed, then success is returned (*optional* and *sufficient* error values are ignored).

If one or more *required* modules fail, then the error value from the first *required* module that failed is returned. If none of the service modules in the stack are designated as *required* or *requisite*, then the PAM framework requires that at least one *optional* or *sufficient* module succeed. If all fail then the error value from the first service module in the stack is returned.

The exception to the above is caused by the *sufficient* flag. If a service module that is designated as *sufficient* succeeds, then the PAM framework immediately returns success to the application (all subsequent service modules, even *required* and *requisite* ones, in the stack are ignored), given that all prior *required* and *requisite* modules have also succeeded. If a prior *required* module failed, then the error value from that module is returned.

If a module does not exist or can not be opened, then the entry is ignored.

5.6.4 Module Path

The *module_path* field specifies the pathname to a shared library object which implements the service functionality. If the pathname is not absolute, it is assumed to be relative to an implementation-defined base directory

The PAM configuration syntax does not dictate either the name or the location of the service specific modules. The convention, however, is the following:

```
/usr/lib/security/pam_<service_name>_<module_name>.<extension>
```

5.6.5 Options

The *options* field is used by the PAM framework layer to pass module specific options to the modules. It is up to the module to parse and interpret the options. This field can be used by the modules to turn on debugging or to pass any module specific parameters such as a TIMEOUT value. It can also be used to support unified login. The options supported by a modules shall be documented by the supplier of the module.

Chapter 6

PAM - Application Program Interface (API)

This chapter presents the functions to be used by callers of the PAM - XSSO application programming interfaces

NAME

pam_acct_mgmt — perform PAM account validation procedures

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_acct_mgmt (
      pam_handle_t        *pamh,
      int                 flags
);
```

DESCRIPTION

The *pam_acct_mgmt*( ) function is called to determine if the current user's account is valid. This includes checking for password and account expiration, as well as verifying access hour restrictions. This function is typically called after the user has been authenticated with *pam_authenticate*( ).

The arguments for *pam_acct_mgmt*( ) are:

pamh (in)
The PAM authentication handle, returned from a previous call to *pam_start*( ).

flags (in)
Flags may be set to:

PAM_SILENT
The account management service should not generate any messages.

PAM_DISALLOW_NULL_AUTHTOK
The account management service should return PAM_NEW_AUTHTOKEN_REQD if the user has a null authentication token

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_ACCT_EXPIRED]
The user account has expired.

[PAM_NEW_AUTHTOKEN_REQD]
New authentication token is required. The user password has aged or expired. PAM service modules return this to request the calling application to immediately prompt the user for a new password.

[PAM_USER_UNKNOWN]
The user is unknown to the underlying account management module.

[PAM_OPEN_ERR]
Failure when dynamically loading an account management service module

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

[PAM_PERM_DENIED]
Permission to access relevant information is denied

[PAM_AUTHTOK_EXPIRED]
User password has aged or expired. Typically, PAM service modules return this to indicate that a password has been expired for too long.

NAME

pam_authenticate — perform authentication within the PAM framework

SYNOPSIS

```
#include &lt;security/pam_appl.h&gt;

int pam_authenticate (
      pam_handle_t        *pamh,
      int                 flags
);
```

DESCRIPTION

The *pam_authenticate*( ) function is called to authenticate a user. The user is usually required to enter a password or similar authentication token depending upon the authentication service configured within the system. The user in question may have been specified by a prior call to *pam_start*( ) or *pam_set_item*( ). The underlying PAM modules may use the PAM conversation functions to get information about the user.

In the case of an authentication failure due to an incorrect username or password as denoted by the error code PAM_AUTH_ERR or PAM_USER_UNKNOWN, it is the responsibility of the application to retry *pam_authenticate* and to maintain the retry count. An authentication service module may implement an internal retry count and return an error PAM_MAXTRIES if the module does not want the application to retry.

If the PAM framework cannot load the authentication module, then it will return PAM_OPEN_ERR.

For security reasons, the location of the authentication failure is hidden from the user. Thus, if several authentication services are stacked and a single service fails, *pam_authenticate* requires that the user re- authenticate to all the services.

A Null authentication token in the authentication database will result in successful authentication unless PAM_DISALLOW_NULL_AUTHTOK was specified. In such cases, there will not be any prompting for the user to enter an authentication token.

For security reasons, *pam_authenticate()*( ) clears the PAM_AUTHTOK item in the PAM handle prior to returning to the application.

The arguments for *pam_authenticate*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*.

flags (in)
: Flags which determine the actions to be taken on authentication. These may be set to

  PAM_SILENT
  : the authentication service shall not display any messages

  PAM_DISALLOW_NULL_AUTHTOK
  : The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_AUTH_ERR]
There has been an error in authenticating the user. This occurs if the user submits an invalid authentication token, or if the PAM_DISALLOW_NULL_AUTHTOK flag is set and the user submits a NULL authentication token.

[PAM_CRED_INSUFFICIENT]
Cannot access authentication data due to insufficient credentials.

[PAM_AUTHINFO_UNAVAIL]
The underlying authentication service cannot retrieve the authentication information.

[PAM_USER_UNKNOWN]
The user is not known to the authentication module.

[PAM_MAXTRIES]
An authentication service has maintained a retry count which has been reached. No more authentication retries should be attempted.

[PAM_OPEN_ERR]
Failure when dynamically loading one of the authentication service modules

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

[PAM_PERM_DENIED]
Permission denied

NAME

pam_authenticate_secondary — perform authentication to a secondary domain within the PAM framework

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_authenticate_secondary (
      pam_handle_t        *pamh,
      char                *target_username,
      char                *target_module_type,
      char                *target_authn_domain,
      char                *target_supp_data,
      unsigned char       *target_module_authtok,
      int                  flags
);
```

DESCRIPTION

The *pam_authenticate_secondary*( ) function is called to authenticate the *target_username* in the domain specified by *target_authn_domain* independently of the primary user authentication and user session establishment. The caller will typically have previously retrieved the username and authentication token to be used with the target domain by calls to *pam_get_mapped_username*( ) and pam_get_mapped_authtok( ).

If the PAM framework cannot load the authentication module, then it will return PAM_OPEN_ERR.

If PAM_DISALLOW_NULL_AUTHTOK is specified and *target_module_authtok* is NULL then the authentication will fail.

Callers should not assume that the *target_module_authtok* buffer will be cleared upon return from this function.

The arguments for *pam_authenticate_secondary*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*.

target_username (in)
: The username to be authenticated within the target domain. This will generally have been retrieved with a call to pam_get_mapped_username( ).

target_module_type (in)
: The mechanism to be used for the authentication.

target_authn_domain (in)
: The domain within which the secondary authentication is required.

target_supp_data (in)
: Supplementary data to be used by the secondary authentication mechanism.

target_module_authtok (in)
: The authentication data specific to the type of mechanism and the domain within which authentication is required. This will generally have been retrieved with a call to pam_get_mapped_authtok( ).

flags (in)
: Flags which determine the actions to be taken on authentication. These may be set to

PAM_SILENT
: the authentication service shall not display any messages

PAM_DISALLOW_NULL_AUTHTOK
: The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_AUTH_ERR]
: There has been an error in authenticating the user. This occurs if the user submits an invalid authentication token, or if the PAM_DISALLOW_NULL_AUTHTOK flag is set and the user submits a NULL authentication token.

[PAM_CRED_INSUFFICIENT]
: Cannot access authentication data due to insufficient credentials.

[PAM_USER_UNKNOWN]
: The user is not known to the authentication module.

[PAM_OPEN_ERR]
: Failure when dynamically loading the secondary authentication service module

[PAM_SYMBOL_ERR]
: Symbol not found in service module

[PAM_SERVICE_ERR]
: Error in service module

[PAM_SYSTEM_ERR]
: System error

[PAM_BUF_ERR]
: Memory buffer error

[PAM_CONV_ERR]
: Conversation error

[PAM_PERM_DENIED]
: ~~Permission denied~~

NAME

pam_chauthtok — perform password related functions within the PAM framework

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_chauthtok (
      pam_handle_t        *pamh,
      int                 flags
);
```

DESCRIPTION

The *pam_chauthtok*( ) function changes the authentication token associated with a particular user referenced by the authentication handle, *pamh*.

pam_chauthtok( ) performs a preliminary check before attempting to update passwords. This check is performed for each password module in the stack as listed in PAM configuration data. The check may include pinging remote name services to determine if they are available. If *pam_chauthtok*( ) returns PAM_TRY_AGAIN, then the check has failed, and passwords are not updated.

The underlying PAM password modules may use the PAM conversation functions to get relevant information from the user.

Note: That it is possible that the password update succeeds only in some modules.

For security reasons, *pam_chauthtok*( ) clears the PAM_AUTHTOK and PAM_OLDAUTHTOK items in the PAM handle prior to returning to the calling application.

The arguments for *pam_chauthtok*( ) are:

pamh (in)
The PAM authentication handle, returned from a previous call to *pam_start*.

flags (in)
Flags may be set to

PAM_SILENT
Disable messages from the password service

PAM_CHANGE_EXPIRED_AUTHTOK
Specify that only expired passwords should be changed. If this flag is not passed, all password services should update their passwords.

The flag PAM_CHANGE_EXPIRED_AUTHTOK is typically used by a *login* application which has determined that the user's password has aged or expired. Before allowing the user to login, the *login* application may invoke *pam_chauthtok*( ) with this flag to allow the user to update the password. Typically applications such as *passwd* should not use this flag.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_AUTHTOK_ERR]
There has been a failure in updating the authentication token.

[PAM_TRY_AGAIN]
Preliminary check on the password has failed. Try again.

[PAM_AUTHTOK_RECOVERY_ERR]
The authentication information cannot be recovered.

[PAM_AUTHTOK_LOCK_BUSY]
The authentication token lock is busy.

[PAM_AUTHTOK_DISABLE_AGING]
Authentication token aging is disabled.

[PAM_USER_UNKNOWN]
The user is unknown to the password service.

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_OPEN_ERR]
Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_close_session — close an existing user session

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_close_session (
      pam_handle_t       *pamh,
      int                flags
);
```

DESCRIPTION

The *pam_close_session*( ) function informs the PAM framework that the user session previously opened by a call to *pam_open_session* has terminated.

In many instances the *pam_open_session*( ) and *pam_close_session*( ) calls may be made by different processes. For example, in UNIX the *login* process opens a session, while the *init* process closes the session. In this case the, UTMP/WTMP entries may be used to link the call to *pam_close_session*( ) with an earlier call to *pam_open_session*( ). This is possible because UTMP/WTMP entries are uniquely identified by a combination of attributes, including the user login name and device name, which are accessible through the PAM handle, *pamh*. The call to *pam_open_session*( ) should precede UTMP/WTMP entry management and the call to *pam_close_session*( ) should follow UTMP/WTMP exit management.

The arguments for *pam_close_session*( ) are:

pamh (in)
: The PAM authentication handle, which has been returned from a previous call to *pam_start*( ).

flags (in)
: Flags may be set to PAM_SILENT to disable messages from the session service.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_SESSION_ERR]
: There has been a failure in creating or removing and entry for the specified session.

[PAM_OPEN_ERR]
: Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
: Symbol not found in service module

[PAM_SERVICE_ERR]
: Error in service module

[PAM_SYSTEM_ERR]
: System error

[PAM_BUF_ERR]
: Memory buffer error

[PAM_CONV_ERR]
: Conversation error

PAM_PERM_DENIED
Permission denied

NAME

pam_end — terminates the PAM transaction

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_end (
      pam_handle_t        *pamh,
      int                 status,
);
```

DESCRIPTION

The *pam_end*( ) function terminates the PAM transaction referred to by *pamh*. The function frees any storage allocated by the PAM modules. *status* is passed to the *cleanup*( ) function stored within the pam handle, *pamh*, and is used to determine what module specific state must be purged. A cleanup function is attached to the handle by the underlying PAM modules through a call to *pam_set_data*( ).

The arguments for *pam_end*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*.

status (in)
Used to determine the module specific state which needs to be purged. This is typically the status of the last PAM call.

RETURN VALUE

One of the following PAM return codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

NAME

pam_get_data — get module information

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_get_data (
      pam_handle_t        *pamh,
      const char          *module_data_name,
      void                **data
);
```

DESCRIPTION

The *pam_get_data*( ) function is used by the PAM modules to retrieve module specific information from the PAM handle, *pamh*, for the *module_data_name* supplied. The *data* argument is assigned the address of the requested data. This data should not be freed by the caller; it will be freed by the cleanup function that was specified in the call to pam_set_data when the pam_end function is called.

The arguments for *pam_get_data*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*( ), from which the data are retrieved.

module_data_name (in)
The name identifying the module data to be retrieved. This should be unique across all services.

data (out)
The data retrieved from *pamh* for *module_data_name* supplied.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_NO_MODULE_DATA]
No module specific data are present.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

NAME

pam_getenv — retrieve the value of a PAM environment variable

SYNOPSIS

```
#include <security/pam_appl.h>

char* pam_getenv (
       pam_handle_t        *pamh,
       const char          *name,
);
```

DESCRIPTION

The *pam_getenv*( ) function returns the value of the environment variable specified by *name*.

The PAM library module will allocate memory for the returned value. The calling application is responsible for freeing that memory. The arguments for *pam_getenv*( ) are:

pamh (in)
: The PAM authentication handle, obtained from a previous call to *pam_start*.

name (in)
: Name of the environment variable to be retrieved.

RETURN VALUE

pam_getenv( ) returns the value of the specified variable. If the variable does not exist, a NULL pointer is returned.

NAME

pam_getenvlist — returns a list of all the PAM environment variables

SYNOPSIS

```
#include <security/pam_appl.h>

char** pam_getenvlist (
       pam_handle_t        *pamh,
       );
```

DESCRIPTION

The *pam_getenvlist*( ) function returns a pointer to a list of all the PAM environment variables.

The PAM library module allocates memory for the returned value. The calling calling application is responsible for freeing this memory.

The arguments for *pam_getenvlist*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*( ).

RETURN VALUE

The *pam_getenvlist*( ) returns a pointer to a list of all the PAM environment variables. If no values are set, a NULL pointer is returned.

NAME

pam_get_item — get PAM information

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_get_item (
      pam_handle_t        *pamh,
      int                 item_type,
      void                **item
);
```

DESCRIPTION

The *pam_get_item*( ) function returns to the caller the PAM information for the *item_type* supplied. *item* is assigned the address of the requested item. The data within the *item* is valid until it is modified by a subsequent call to *pam_set_item*( ). If the item has not been previously set, a NULL pointer is returned.

An *item* retrieved by *pam_get_item*( ) should not be modified or freed. It will be released by *pam_end*( ).

The arguments for *pam_get_item*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*.

item_type (in)
The item type for which the PAM information is requested. This may be:

PAM_SERVICE
The service name

PAM_USER
The user name

PAM_AUTHOK
The user authentication token

PAM_OLDAUTHOK
The old user authentication token

PAM_TTY
The tty name

PAM_RHOST
The remote host name

PAM_RUSER
The remote user name

PAM_CONV
The *pam_conv* structure

PAM_USER_PROMPT
The default prompt used by *pam_get_user*( ).

The item types **PAM_AUTHTOK** and **PAM_OLDAUTHTOK** are available only to the PAM service modules for security reasons. The authentication module, account module, and session management module should treat **PAM_AUTHTOK** as the current authentication token, and should ignore **PAM_OLDAUTHTOK**. The password management module should treat **PAM_OLDAUTHTOK** as the current authentication token and **PAM_AUTHTOK** as the new authentication token.

item (out)
The address of a pointer into which is returned the address of the object requested.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

NAME

pam_get_mapped_authtok — get mapped password for the user

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_get_mapped_authtok (
    pam_handle_t          *pamh,
    const char            *target_module_username,
    const char            *target_module_type,
    const char            *target_authn_domain,
    size_t                *target_authtok_len,
    unsigned char         **target_module_authtok
);
```

DESCRIPTION

The *pam_get_mapped_authtok*( ) function is used to obtain a password for the given user. Any authorisation data required by the implementation of this interface must be present in the PAM handle. The function checks the authorisation data provided in the PAM handle to ensure that the caller is authorised to retrieve the password for the *target_module_username*.

The caller should clear memory containing the returned password immediately after using the password.

The arguments for *pam_get_mapped_authtok*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*.

target_module_username (in)
: The target username used for the mapping.

target_module_type (in)
: The target authentication type, for example, UNIX

target_authn_domain (in)
: The target domain, e.g., the UNIX hostname. A NULL value may be passed to allow a default target domain to be used.

target_authtok_len (out)
: The length of the target password.

target_module_authtok (out)
: The target password.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_USER_UNKNOWN]
: The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
: The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
: The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_OPEN_ERR]
Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
Symbol not found

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_get_mapped_username — get valid matched identity in new domain

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_get_mapped_username (
      pam_handle_t        *pamh,
      const char          *src_username,
      const char          *src_module_type,
      const char          *src_authn_domain,
      const char          *target_module_type,
      const char          *target_authn_domain,
      char                **target_module_username,
);
```

DESCRIPTION

The *pam_get_mapped_username*( ) function is used to obtain a valid identity in a new domain that matches the input identity. The *target_module_type* and *target_authn_domain* are used to query the mapping database and extract the *target_username*.

The arguments for *pam_get_mapped_username*( ) are:

pamh (in)
The PAM authentication handle, which has been returned from a previous call to *pam_start*.

src_username (in,out)
The source username used for the mapping. It this is NULL, then the value is obtained from the *pam_hamdle*.

src_module_type (in)
The source authentication type, for example, DCE

src_authn_domain (in)
The source domain, eg., the DCE cell name

target_module_type (in)
The target authentication type, for example, UNIX

target_authn_domain (in)
The target domain, for example, UNIX hostname

target_module_username (out)
The target username which matches the input *src_username*.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_USER_UNKNOWN]
The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
The mapping service does not support this module.

[PAM_DOMAIN_UNKNOWN]
The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_OPEN_ERR]
Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
Symbol not found

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_get_user — retrieve user name

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_get_user (
      pam_handle_t        *pamh,
      char                **user,
      const char          *prompt
);
```

DESCRIPTION

The *pam_get_user*( ) function is used by PAM service modules to retrieve the current user name from the *pamh* handle. If the user name has not been set via *pam_start*( ) or *pam_set_item*( ), then the PAM conversation function will be used to prompt the user for the user name with the string *prompt*. If *prompt* is NULL, then *pam_get_item*( ) is called and the value of PAM_USER_PROMPT is used for prompting. If the value of PAM_USER_PROMPT is NULL, the following default prompt is used:

```
Please enter user name:
```

After the user name is gathered by the conversation function, *pam_set_item*( ) is used to set the value of PAM_USER.

By convention, applications that need to prompt for a user name should call *pam_set_item*( ) and set the value of PAM_USER_PROMPT before calling *pam_authenticate*( ). The service module's *pam_sm_authenticate*( ) function will then call *pam_get_user*( ) to prompt for the user name. Note that certain PAM service modules (such as a smart card module) may override the value of PAM_USER_PROMPT and pass in their own prompt.

Applications that call *pam_authenticate*( ) multiple times should set the value of PAM_USER to NULL with *pam_set_item*( ) before calling *pam_authenticate*( ) if they want the user to be prompted for a new user name each time.

The value of *user* retrieved by *pam_get_user*( ) should not be modified or freed. The item will be released by *pam_end*( ).

The arguments for *pam_get_user*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*( ).

user (out)
The user name returned from *pamh*.

prompt (in)
The prompt to be used if the conversation function needs to prompt the user for a user name.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_CONV_ERR]
Conversation failure.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_open_session — open a user session

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_open_session (
      pam_handle_t        *pamh,
      int                 flags
);
```

DESCRIPTION

The *pam_open_session*( ) function opens a new session for a user previously authenticated with a call to *pam_authenticate*( ).

If successful, the function returns [PAM_SUCCESS].

The arguments for *pam_open_session*( ) are:

pamh (in)
The PAM handle, which has been returned from a previous call to *pam_start*.

flags (in)
Flags may be set to PAM_SILENT to disable messages from the session service.

In many instances the *pam_open_session*( ) and *pam_close_session*( ) calls may be made by different processes. For example, in UNIX the *login* process opens a session, while the *init* process closes the session. In this case the, UTMP/WTMP entries may be used to link the call to *pam_close_session*( ) with an earlier call to *pam_open_session*( ). This is possible because UTMP/WTMP entries are uniquely identified by a combination of attributes, including the user login name and device name, which are accessible through the PAM handle, *pamh*. The call to *pam_open_session*( ) should precede UTMP/WTMP entry management and the call to *pam_close_session*( ) should follow UTMP/WTMP exit management.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SESSION_ERR]
There has been a failure in creating or removing an entry for the specified session.

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_OPEN_ERR]
Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_putenv — set the value of an environment variable

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_putenv (
      pam_handle_t        *pamh,
      const char          *namevalue,
);
```

DESCRIPTION

The function *pam_putenv*( ) is used by the PAM service modules to set the value of the environment variable defined by *namevalue*.

The arguments for *pam_putenv*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*.

namevalue (in)
Name and value of the environment variable to be set. It should be in the form "NAME=value", for example, "SHELL=/bin/sh"

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SYSTEM_ERR]
The environment variable could not be set.

[PAM_BUF_ERR]
Memory buffer error

ERROR

No other errors are defined.

NAME

pam_setcred — modify/delete user credentials for an authentication service

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_setcred (
       pam_handle_t        *pamh,
       int                 flags
);
```

DESCRIPTION

The *pam_setcred*( ) function is used to establish, modify, or delete the credentials of the current user associated with the authentication handle, *pamh*.

The arguments for *pam_setcred*( ) are:

pamh (in)
: The PAM authentication handle, which has been returned from a previous call to *pam_start*.

flags (in)
: Flags may be set to one of the following:

  PAM_SILENT
  : to disable messages from the authentication service

  PAM_ESTABLISH_CRED
  : to set user credentials. This is the default value.

  PAM_DELETE_CRED
  : to delete user credentials associated with the authentication service

  PAM_REINITIALISE_CRED
  : to reinitialise user credentials

  PAM_REFRESH_CRED
  : to extend the lifetime of the user credentials

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_CRED_UNAVAIL]
: The authentication service cannot retrieve the user credentials.

[PAM_CRED_EXPIRED]
: The user credentials have expired.

[PAM_USER_UNKNOWN]
: The user is unknown to the service.

[PAM_CRED_ERR]
: Failure in setting user credentials.

[PAM_PERM_DENIED]
: The caller does not possess the required authority.

[PAM_OPEN_ERR]
: Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
Symbol not found

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_set_data — set module information

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_set_data (
      pam_handle_t        *pamh,
      const char          *module_data_name,
      void                *data,
      void                *(cleanup) (pam_handle_t *pamh,
                          void *data, int pam_end_status)
);
```

DESCRIPTION

The *pam_set_data*( ) function allows the PAM modules to set and update module specific information as needed. The module specific information is stored within the PAM handle, *pamh*. The *module_data_name* uniquely identifies the data, and the *data* argument represents the data.

The *cleanup*( ) function is used to free any memory used by the data after it is no longer needed and is invoked by *pam_end*( ). If *pam_set_data*( ) is called and the module data already exists under the same *module_data_name*, then the existing data are replaced by the new data and the existing *cleanup* function is replaced by the new *cleanup* function.

The arguments for *pam_set_data*( ) are:

pamh (in)
: The PAM authentication handle, obtained from a previous call to *pam_start*( ). It is used to store the module specific data set by the function.

module_data_name (in)
: The name identifying the module data to be set. This should be unique across all services.

data (in)
: The data to be set for *module_data_name* supplied.

cleanup (in)
: The cleanup function to be used by *pam_end* defining what module specific information needs to be purged on termination.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_SYSTEM_ERR]
: System error

[PAM_BUF_ERR]
: Memory buffer error

NAME

pam_set_item — set authentication information

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_set_item (
      pam_handle_t        *pamh,
      int                 item_type,
      void                *item
);
```

DESCRIPTION

The *pam_set_item*( ) function allows the caller to update PAM information as needed. The information is specified by *item_type*.

The function copies the item to an internal storage area allocated by the authentication module. If the item had been previously set, it is overwritten.

The arguments for *pam_set_item*( ) are:

pamh (in)
: The PAM handle, obtained from a previous call to *pam_start*( ).

item_type (in)
: This may be one of:

PAM_SERVICE
: The service name

PAM_USER
: The user name

PAM_AUTHOK
: The user authentication token

PAM_OLDAUTHOK
: The old user authentication token

PAM_TTY
: The tty name

PAM_RHOST
: The remote host name

PAM_RUSER
: The remote user name

PAM_CONV
: The *pam_conv* structure

PAM_USER_PROMPT

: The default prompt used by *pam_get_user*( ).

The authentication module, account module, and session management module should treat **PAM_AUTHTOK** as the current authentication token, and should ignore **PAM_OLDAUTHTOK**. The password management module should treat **PAM_OLDAUTHTOK** as the current authentication token and **PAM_AUTHTOK** as the new authentication token.

item (in)
A pointer to the object to be set or updated.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

NAME

pam_set_mapped_authtok — store the password for the username supplied

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_set_mapped_authtok (
      pam_handle_t        *pamh,
      char                *target_module_username,
      size_t              *target_authtok_len,
      unsigned char       *target_module_authtok,
      char                *target_module_type,
      char                *target_authn_domain,
);
```

DESCRIPTION

The *pam_set_mapped_authtok*( ) function stores the password for the *target_username* supplied.

The arguments for *pam_set_mapped_authtok*( ) are:

pamh (in)
The PAM authentication handle, returned from a previous call to *pam_start*.

target_module_username (in)
The target username for which the password is to be stored.

target_authtok_len (in)
The length of the password to be stored.

target_module_authtok (in)
The password to be stored.

target_module_type (in)
The target authentication type, for example, UNIX

target_authn_domain (in)
The target domain, eg., the UNIX hostname

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_USER_UNKNOWN]
The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_OPEN_ERR]
Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_set_mapped_username — set a username

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_set_mapped_username (
    pam_handle_t          *pamh,
    char                  *src_username,
    char                  *src_module_type,
    char                  *src_authn_domain,
    char                  *target_module_username,
    char                  *target_module_type,
    char                  *target_authn_domain,
);
```

DESCRIPTION

The *pam_set_mapped_username*( ) function stores a username using the *target_module_type* and *target_authn_domain* parameters supplied.

The arguments for *pam_set_mapped_username*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*( ).

target_module_username (in)
: The target username to be stored.

target_module_type (in)
: The target authentication type, for example, UNIX

target_authn_domain (in)
: The target domain, eg., the UNIX hostname

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_USER_UNKNOWN]
: The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
: The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
: The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
: The mapping service failed in reading/writing data.

[PAM_PERM_DENIED]
: The caller does not possess the required authority.

[PAM_OPEN_ERR]
: Failure when dynamically loading a service module

[PAM_SYMBOL_ERR]
: Symbol not found in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_acct_mgmt — Service provider implementation for pam_acct_mgmt

SYNOPSIS

```
#include &lt;security/pam_appl.h&gt;
#include &lt;security/pam_modules.h&gt;

int pam_sm_acct_mgmt (
      pam_handle_t        *pamh,
      int                 flags,
      int                 argc,
      const char          **argv
);
```

DESCRIPTION

In response to a call to *pam_acct_mgmt*( ), the PAM framework calls *pam_sm_acct_mgmt*( ) from the modules listed in the PAM configuration. The authentication provider supplies the back-end functionality for this interface function.

The function *pam_sm_acct_mgmt*( ), is called to etermine if the current user's account is valid. This includes checking for password and account expiration, as well as verifying access hour restrictions. This function is typically called after the user has been authenticated with *pam_authenticate*( ).

The arguments for *pam_acct_mgmt*( ) are:

pamh (in)
The PAM authentication handle, returned from a previous call to *pam_start*( ).

flags (in)
Flags may be set to:

PAM_SILENT
The account management service should not generate any messages.

PAM_DISALLOW_NULL_AUTHTOK
The account management service should return PAM_NEW_AUTHTOKEN_REQD if the user has a null authentication token

argc (in)
The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
specifies the module options, which are interpreted and processed by the authentication module. Please refer to the specific module manual pages for the various available *options.* If any unknown option is passed in, the module should log the error and ignore the option.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_ACCT_EXPIRED]
The user account has expired.

[PAM_NEW_AUTHTOKEN_REQD]
New authentication token is required. The user password has aged or expired. PAM service

modules return this to request the calling application to immediately prompt the user for a new password.

[PAM_USER_UNKNOWN]
The user is unknown to the underlying account management module.

[PAM_OPEN_ERR]
Failure when dynamically loading an account management service module

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation failure.

[PAM_PERM_DENIED]
Permission to access relevant information is denied

[PAM_AUTHTOK_EXPIRED]
User password has aged or expired. typically, PAM service modules return this to indicate that a password has been expired for too long.

NAME

pam_sm_authenticate — Service provider implementation for pam_authenticate

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_authenticate(
      pam_handle_t *      pamh,
      int                 flags,
      int                 argc,
      const char          **argv
);
```

DESCRIPTION

In response to a call to *pam_authenticate*( ), the PAM framework calls *pam_sm_authenticate*( ) from the modules listed in the PAM configuration. The authentication provider supplies the back-end functionality for this interface function.

The function, *pam_sm_authenticate*( ), is called to verify the identity of the current user. The user is usually required to enter a password or similar authentication token depending upon the authentication scheme configured within the system. The user in question is typically specified by a prior call to *pam_start*( ), and is referenced by the authentication handle, *pamh.*

If the user is unknown to the authentication service, the service module should mask this error and continue to prompt the user for a password. It should then return the error, PAM_USER_UNKNOWN.

Before returning, *pam_sm_authenticate*( ) should call *pam_get_item*( ) and retrieve PAM_AUTHTOK. If it has not been set before (ie. the value is NULL), *pam_sm_authenticate*( ) should set it to the password entered by the user using *pam_set_item*( ).

An authentication module may save the authentication status (success or reason for failure) as state in the authentication handle using *pam_set_data*( ). This information is intended for use by *pam_setcred*( ).

Note: Modules should not retry the authentication in the event of a failure. Applications handle authentication retries. To limit the number of retries, modules may maintain an internal retry count and return a PAM_MAXTRIES error.

The arguments for *pam_sm_authenticate*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*( ).

flags (in)
The following flags may be passed in to *pam_sm_authenticate*( ):

PAM_SILENT
The authentication service should not generate any messages

PAM_DISALLOW_NULL_AUTHTOK
The authentication service should return PAM_AUTH_ERR if the user has a null authentication token

argc (in)
The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
specifies the module options, which are interpreted and processed by the authentication module. Please refer to the specific module manual pages for the various available *options.* If any unknown option is passed in, the module should log the error and ignore the option.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_AUTH_ERR]
The user could not be authenticated.

[PAM_USER_UNKNOWN]
No account for the present user.

[PAM_CRED_INSUFFICIENT]
Cannot access authentication data because of insufficient credentials.

[PAM_AUTHINFO_UNAVAIL]
Cannot retrieve authentication information

[PAM_IGNORE]
Ignore underlying authentication module regardless of whether the control flag is *required*, *optional*, or *sufficient*.

[PAM_CONV_ERR]
Conversation failure.

[PAM_SERVICE_ERR]
Error in underlying service module.

[PAM_MAXTRIES]
The module can return this error to limit the number of retries.

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

NAME

pam_sm_authenticate_secondary — Service provider interface for pam_authenticate_secondary

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_authenticate_secondary (
      pam_handle_t        *pamh,
      char                *target_username,
      char                *target_module_type,
      char                *target_authn_domain,
      char                *target_supp_data,
      unsigned char       *target_module_authtok,
      int                  flags
);
```

DESCRIPTION

In response to a call to *pam_authenticate_secondary*( ), the PAM framework calls *pam_sm_authenticate_secondary*( ) from the modules listed in the PAM configuration. The authentication provider supplies the back-end functionality for this interface function.

The function, *pam_sm_authenticate_secondary*( ), is called to verify the identity of the current user to a further domain.

If PAM_DISALLOW_NULL_AUTHTOK is specified and *target_module_authtok* is NULL then the authentication will fail.

The arguments for *pam_sm_authenticate_secondary*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*.

target_username (in)
: The username to be authenticated within the target domain.

target_module_type (in)
: The mechanism to be used for the authentication.

target_authn_domain (in)
: The domain within which the secondary authentication is required.

target_supp_data (in)
: Supplementary data to be used by the secondary authentication mechanism.

target_module_authtok (in)
: The authentication data specific to the type of mechanism and the domain within which authentication is required. This will generally have been retrieved with a call to pam_get_mapped_authtok( ).

flags (in)
: Flags which determine the actions to be taken on authentication. These may be set to

  PAM_SILENT
  : the authentication service shall not display any messages

  PAM_DISALLOW_NULL_AUTHTOK
  : The authentication service should return PAM_AUTH_ERROR if the user has a null authentication token.

RETURN VALUE

One of the following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_AUTH_ERR]
There has been an error in authenticating the user. This occurs if the user submits an invalid authentication token, or if the PAM_DISALLOW_NULL_AUTHTOK flag is set and the user submits a NULL authentication token.

[PAM_CRED_INSUFFICIENT]
Cannot access authentication data due to insufficient credentials.

[PAM_USER_UNKNOWN]
The user is not known to the authentication module.

[PAM_SYMBOL_ERR]
Symbol not found in service module

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

[PAM_PERM_DENIED]
Permission denied

NAME

pam_sm_chauthtok — Service provider implementation for pam_chauthtok

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_chauthtok(
      pam_handle_t  *pamh,
      const int     flags,
      int           argc,
      const char    **argv
);
```

DESCRIPTION

In response to a call to *pam_chauthtok*( ) the PAM framework calls *pam_sm_chauthtok*( ) from the modules listed in the PAM configuration. The password management provider supplies the back-end functionality for this interface function.

pam_sm_chauthtok( ) changes the authentication token associated with a particular user referenced by the authentication handle, *pamh.*

Upon successful completion of the call, the authentication token of the user will be ready for change or will be changed (depending upon the flag) in accordance with the authentication scheme configured within the system.

It is the responsibility of *pam_sm_chauthtok*( ) to determine if the new password meets certain strength requirements. *pam_sm_chauthtok*( ) may continue to re-prompt the user (for a limited number of times) using the conversation functions for a new password until the password entered meets the strength requirements.

Before returning, *pam_sm_chauthtok*( ) should call *pam_get_item*( ) and retrieve both PAM_AUTHTOK and PAM_OLDAUTHTOK. If both are NULL, *pam_sm_chauthtok*( ) should set them to the new and old passwords as entered by the user.

Note that the framework invokes the password services twice. The first time the modules are invoked with the flag, PAM_PRELIM_CHECK. During this stage, the password modules should only perform preliminary checks (ping remote name services to see if they are ready for updates, for example). If a password module detects a transient error (remote name service temporarily down, for example) it should return PAM_TRY_AGAIN to the PAM framework, which will immediately return the error back to the application. If all password modules pass the preliminary check, the PAM framework invokes the password services again with the flag, PAM_UPDATE_AUTHTOK. During this stage, each password module should proceed to update the appropriate password. Any error will again be reported back to application.

If a service module receives the flag, PAM_CHANGE_EXPIRED_AUTHTOK, it should check whether the password has aged or expired. If the password has aged or expired, then the service module should proceed to update the password. If the status indicates that the password has not yet aged/expired, then the password module should return PAM_IGNORE.

If a user's password has aged or expired, a PAM account module could save this information as state in the authentication handle, *pamh*, using *pam_set_data*( ). The related password management module could retrieve this information using *pam_get_data*( ) to determine whether or not it should prompt the user to update the password for this particular module.

The arguments for *pam_sm_chauthtok*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*( ).

flags (in)
The following flag may be passed in to *pam_sm_chauthtok*( ):

PAM_SILENT
The password service should not generate any messages

PAM_CHANGE_EXPIRED_AUTHTOK
The password service should only update those passwords that have aged. If this flag is not passed, the password service should update all passwords.

PAM_PRELIM_CHECK
The password service should only perform preliminary checks. No passwords should be updated.

PAM_UPDATE_AUTHTOK
The password service should update passwords

Note that PAM_PRELIM_CHECK and PAM_UPDATE_AUTHTOK can not be set at the same time.

argc (in)
The *argc* argument represents the number of module options passed in from the PAM configuration.

argv (in)
specifies the module options, which are interpreted and processed by the password management module. Please refer to the specific module man pages for the various available *options.*

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_AUTHTOK_ERR]
Authentication token manipulation error.

[PAM_AUTHTOK_RECOVERY_ERR]
Old authentication token cannot be retrieved.

[PAM_AUTHTOK_LOCK_BUSY]
The authentication token lock is busy.

[PAM_AUTHTOK_DISABLE_AGING]
Authentication token again disabled

[PAM_USER_UNKNOWN]
User unknown to password service

[PAM_TRY_AGAIN]
Preliminary check by password service failed.

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_sm_close_session — Service provider implementation for pam_close_session

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_close_session(
      pam_handle_t        *pamh,
      int                 flags,
      int                 argc,
      const char **       argv
);
```

DESCRIPTION

In response to a call to *pam_close_session*( ) the PAM framework calls *pam_sm_close_session*( ) from the modules listed in the pam.conf file. The session management provider supplies the back-end functionality for this interface function.

pam_sm_close_session( ) is called to terminate session management.

The arguments for *pam_sm_close_session*( ) are:

pamh (in)
The PAM authentication handle, obtained from a previous call to *pam_start*( ).

flags (in)
The following flag may be set in the *flags* field:

PAM_SILENT
Session service should not generate any messages

argc (in)
The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
specifies the module options, which are interpreted and processed by the session management service. If an unknown option is passed in, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SESSION_ERR]
Can not make/remove an entry for the specified session

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_sm_get_mapped_authtok — get password for username

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_get_mapped_authtok (
    pam_handle_t           *pamh,
    char                   *target_module_username,
    char                   *arget_module_type,
    char                   *arget_authn_domain,
    size_t                 *target_authtok_len,
    unsigned char          **arget_module_authtok,
    int                           argc,
    const char **          argv
);
```

DESCRIPTION

The *pam_sm_get_mapped_authtok*( ) function is used to obtain a password for the username supplied. Any authorisation data required by the implementation of this interface must be present in the PAM handle. The function checks the authorisation data provided in the PAM handle to ensure that the caller is authorised to retrieve the password for the *target_module_username*.

The caller should clear memory containing the returned password immediately after using the password.

The arguments for *pam_sm_get_mapped_authtok*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*( ).

target_module_username (in)
: The target username used for the mapping.

target_module_type (in)
: The target authentication type, for example, UNIX

target_authn_domain (in)
: The target domain, eg., the UNIX hostname

target_authtok_len (out)
: The length of the target password.

target_module_authtok (out)
: The target password.

argc (in)
: The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
: specifies the module options, which are interpreted and processed by the mapping module. If an unknown option is passed in, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_USER_UNKNOWN]
The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

ERROR

No other errors are defined.

NAME

pam_sm_get_mapped_username — get valid matched identity in new domain

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_get_mapped_username (
      pam_handle_t        *pamh,
      char                *src_username,
      char                *src_module_type,
      char                *src_authn_domain,
      char                *target_module_type,
      char                *target_authn_domain,
      char                **target_module_username,
      int                 argc,
      const char **       argv
);
```

DESCRIPTION

The *pam_sm_get_mapped_username* function is used to obtain a valid identity in a new domain that matches the input identity. The *target_module_type* and *target_authn_domain* are used to query the mapping database and extract the *target_username*.

The arguments for *pam_sm_get_mapped_username*( ) are:

pamh (in)
The PAM authentication handle, which has been returned from a previous call to *pam_start*( ).

src_username (in,out)
The source username used for the mapping. It this is NULL, then the value is obtained from the *pam_hamdle*. If a zero length string is specified, it is used to query the mapping service and the value is returned if found.

src_module_type (in)
The source authentication type, for example, DCE

src_authn_domain (in)
The source domain, eg., the DCE cell name

target_module_type (in)
The target authentication type, for example, UNIX

target_authn_domain (in)
The target domain, for example, UNIX hostname

target_module_username (out)
The target username which matches the input *src_username*.

argc (in)
The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
specifies the module options, which are interpreted and processed by the mapping module. If an unknown option is passed in, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_USER_UNKNOWN]
The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_sm_open_session — Service provider implementation for pam_open_session

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_open_session(
      pam_handle_t        *pamh,
      int                 flags,
      int                 argc,
      const char **       argv
);
```

DESCRIPTION

In response to a call to *pam_open_session*( ) the PAM framework calls *pam_sm_open_session*( ) from the modules listed in the PAM configuration. The session management provider supplies the back-end functionality for this interface function.

pam_sm_open_session( ) is called to initiate session management.

The arguments for *pam_sm_open_session*( ) are:

pamh (in)
: The PAM authentication handle, obtained from a previous call to *pam_start*.

flags (in)
: The following flag may be set in the *flags* field:

PAM_SILENT
: Session service should not generate any messages

argc (in)
: The *argc* argument represents the number of module options passed in from the PAM configuration.

argv (in)
: specifies the module options, which are interpreted and processed by the session management service. If an unknown option is passed in, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_SESSION_ERR]
: Can not make/remove an entry for the specified session

[PAM_IGNORE]
: Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
: The caller does not possess the required authority.

[PAM_SERVICE_ERR]
: Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_sm_set_mapped_authtok — store the password for the username supplied

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_set_mapped_authtok (
      pam_handle_t        *pamh,
      char                *target_module_username,
      size_t              *target_authtok_len,
      unsigned char       *target_module_authtok,
      char                *target_module_type,
      char                *target_authn_domain,
      int                 argc,
      const char **       argv
);
```

DESCRIPTION

The *pam_sm_set_mapped_authtok*( ) function stores the password for the *target_username* supplied.

The arguments for *pam_sm_set_mapped_authtok*( ) are:

pamh (in)
: The PAM authentication handle, returned from a previous call to *pam_start*( ).

target_module_username (in)
: The target username for which the password is to be stored.

target_authtok_len (in)
: The length of the password to be stored.

target_module_authtok (in)
: The password to be stored.

target_module_type (in)
: The target authentication type, for example, UNIX

target_authn_domain (in)
: The target domain, eg., the UNIX hostname

argc (in)
: The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
: specifies the module options, which are interpreted and processed by the mapping module. If an unknown option is passed in, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
: Successful completion.

[PAM_USER_UNKNOWN]
: The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_sm_set_mapped_username — set a username

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_set_mapped_username (
    pam_handle_t          *pamh,
    char                  *target_module_username,
    char                  *arget_module_type,
    char                  *arget_authn_domain,
    int                          argc,
    const char **         argv
);
```

DESCRIPTION

The *pam_sm_set_mapped_username* function stores a username using the *target_module_type* and *target_authn_domain* parameters supplied.

The arguments for *pam_sm_set_mapped_username*( ) are:

pamh (in)
The PAM authentication handle, returned from a previous call to *pam_start*( ).

target_module_username (in)
The target username to be stored.

target_module_type (in)
The target authentication type, for example, UNIX

target_authn_domain (in)
The target domain, eg., the UNIX hostname

argc (in)
The *argc* argument represents the number of module options defined in the PAM configuration.

argv (in)
specifies the module options, which are interpreted and processed by the mapping module. If an unknown option is passed in, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_USER_UNKNOWN]
The username supplied is not recognised.

[PAM_MODULE_UNKNOWN]
The mapping service does not support this module type.

[PAM_DOMAIN_UNKNOWN]
The mapping service does not support this module's domain.

[PAM_SERVICE_ERR]
The mapping service failed in reading/writing data.

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

NAME

pam_sm_setcred — Service provider implementation for pam_setcred

SYNOPSIS

```
#include <security/pam_appl.h>
#include <security/pam_modules.h>

int pam_sm_setcred(
      pam_handle_t  *pamh,
      int           flags,
      int           argc,
      const char    **argv
);
```

DESCRIPTION

In response to a call to *pam_set_cred*(), the PAM framework calls *pam_sm_setcred*() from the modules listed in the PAM configuration. The authentication provider supplies the back-end functionality for this interface function.

pam_sm_setcred() is called to set the credentials of the current user associated with the authentication handle, *pamh.*

The authentication status (success or reason for failure) is typically saved as module-specific state in the authentication handle by the authentication module. The status should be retrieved using *pam_get_data*( ), and used to determine if user credentials should be set.

The arguments for *pam_sm_setcred*( ) are:

pamh (in)
: The PAM authentication handle, obtained from a previous call to *pam_start*.

flags (in)
: The following flags may be set in the *flags* field. Note that the first four flags are mutually exclusive:

PAM_ESTABLISH_CRED
: Set user credentials for the authentication service

PAM_DELETE_CRED
: Delete user credentials associated with the authentication service

PAM_REINITIALIZE_CRED
: Reinitialise user credentials

PAM_REFRESH_CRED
: Extend lifetime of user credentials

PAM_SILENT
: Authentication service should not generate messages

If none of these flags are set, PAM_ESTABLISH_CRED is used as the default.

argc (in)
: The *argc* argument represents the number of module options passed in from the PAM configuration.

argv (in)
: specifies the module options, which are interpreted and processed by the authentication service. If an unknown option is to the module, an error should be logged and the option ignored.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_CRED_UNAVAIL]
Underlying authentication service cannot retrieve user credentials.

[PAM_CRED_EXPIRED]
User credentials have expired.

[PAM_USER_UNKNOWN]
User unknown to authentication service.

[PAM_CRED_ERR]
Failure in setting user credentials.

[PAM_IGNORE]
Ignore underlying session module regardless of whether the control flag is *required, optional* or *sufficient*

[PAM_PERM_DENIED]
The caller does not possess the required authority.

[PAM_SERVICE_ERR]
Error in service module

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

[PAM_CONV_ERR]
Conversation error

ERROR

No other errors are defined.

NAME

pam_start — initiate a PAM transaction

SYNOPSIS

```
#include <security/pam_appl.h>

int pam_start (
      const char          *service,
      const har           *user,
      const struct pam_conv      *pam_conv,
      pam_handle_t        **pamh
);
```

DESCRIPTION

The *pam_start*( ) function initiates a PAM transaction. On successful completion, *pamh* refers to a PAM handle for use with subsequent calls to the authentication library.

The arguments for *pam_start*( ) are:

service (in)
: The name of the current service

user (in)
: The name of the user to be authenticated

pam_conv (in)
: The address of the conversation structure

pamh (out)
: The pam authentication handle is returned. This is then used for all subsequent calls to the authentication library.

The pam_conv structure, *pam_conv,* contains the address of the conversation function provided by the application. The underlying **PAM** service module invokes this function to output information to and retrieve input from the user. The *pam_conv* structure has the following entries:

```
struct pam_conv {
        int     (*conv)();       /* Conversation function */
        void    *appdata_ptr;    /* Application data */
};
      where

        int conv(int num_msg,
                const struct pam_message **msg, struct pam_response **resp,
                void *appdata_ptr);
```

The function *conv*( ) is called by a service module to hold a **PAM** conversation with the application or user. For window applications, the application can create a new pop-up window to be used by the interaction.

The parameter *num_msg* is the number of messages associated with the call. The parameter *msg* is a pointer to an array of length *num_msg* of the *pam_message* structure.

The structure *pam_message* is used to pass prompt, error message, or any text information from the authentication service to the application or user. It is the responsibility of the **PAM** service modules to localise the messages. The memory used by *pam_message* has to be allocated and freed by the PAM modules. The *pam_message* structure has the following entries:

```
struct pam_message{
        int     msg_style;
        char    *msg;
};
```

The message style, *msg_style,* can be set to one of the following values:

PAM_PROMPT_ECHO_OFF
: prompt user, disabling echoing of response

PAM_PROMPT_ECHO_ON
: prompt user, enabling echoing of response

PAM_ERROR_MSG
: print error message

PAM_TEXT_INFO
: print general text information

The maximum size of the message and the response string is PAM_MAX_MSG_SIZE defined in **<security/pam_appl.h>**.

The structure *pam_response* is used by the authentication service to get the user's response back from the application or user. The storage used by *pam_response* has to be allocated by the application and freed by the PAM modules. The *pam_response* structure has the following entries:

```
struct pam_response{
        char    *resp;
        int     resp_retcode;    /* currently not used, should be set to 0 */
};
```

It is the responsibility of the conversation function to strip off newline characters for **PAM_PROMPT_ECHO_OFF** and **PAM_PROMPT_ECHO_ON** message styles, and to add newline characters (if appropriate) for **PAM_ERROR_MSG** and **PAM_TEXT_INFO** message styles.

appdata_ptr is an application data pointer which is passed by the application to the PAM service modules. Since the PAM modules pass it back through the conversation function, the applications can use this pointer to point to any application-specific data.

pam_end( ) is called to terminate the authentication transaction identified by *pamh* and to free any storage area allocated by the authentication module. The argument, *status*, is passed to the *cleanup*( ) function stored within the pam handle, and is used to determine what module specific state must be purged. A cleanup function is attached to the handle by the underlying **PAM** modules through a call to *pam_set_item*( ) to free module specific data.

RETURN VALUE

The following PAM status codes shall be returned:

[PAM_SUCCESS]
Successful completion.

[PAM_SERVICE_ERR]
Error in underlying service module.

[PAM_SYSTEM_ERR]
System error

[PAM_BUF_ERR]
Memory buffer error

NAME

pam_strerror — get PAM standard error message string

SYNOPSIS

```
#include <security/pam_appl.h>

const char *pam_strerror (
      pam_handle_t       *pamh,
      int                error_number
);
```

DESCRIPTION

The *pam_strerror*( ) function maps the PAM error number in *error_number* to a PAM error message and returns the error message. The application should not free or modify the string returned. The function returns a NULL if cannot map *error_number* to a string.

The arguments for *pam_strerror*( ) are:

pamh (in)
: The PAM authentication handle.

error_number (in)
: The number of the error for which the error message is required.

RETURN VALUE

If successful, the string mapped to *error_number* is returned.

If *error_number* cannot be mapped to a string, NULL is returned.

Appendix A

Example Header Files

This appendix provides an illustration of the header files required for a PAM implementation.

A.1 PAM_APPL.H

```
#define  PAM_SUCCESS                  0      /* Normal function return */
#define  PAM_OPEN_ERR                 1      /* Failure in loading service module*/
#define  PAM_SYMBOL_ERR               2      /* Symbol not found */
#define  PAM_SERVICE_ERR              3      /* Error in underlying service module */
#define  PAM_SYSTEM_ERR               4      /* System error */
#define  PAM_BUF_ERR                  5      /* Memory buffer error */
#define  PAM_CONV_ERR                 6      /* Conversation failure */
#define  PAM_PERM_DENIED              7      /* Permission denied */
#define  PAM_MAXTRIES                 8      /* Maximum number of tries exceeded */
#define  PAM_AUTH_ERR                 9      /* Authentication failure */
#define  PAM_NEW_AUTHTOK_REQD         10     /* Get new auth token from the user */
#define  PAM_CRED_INSUFFICIENT        11     /* can not access auth data b/c */
                                             /* of insufficient credentials  */
#define  PAM_AUTHINFO_UNAVAIL         12     /* Can not retrieve auth information */
#define  PAM_USER_UNKNOWN             13     /* No account present for user */
#define  PAM_CRED_UNAVAIL             14     /* can not retrieve user credentials */
#define  PAM_CRED_EXPIRED             15     /* user credentials expired */
#define  PAM_CRED_ERR                 16     /* failure setting user credentials */
#define  PAM_ACCT_EXPIRED             17     /* user account has expired */
#define  PAM_AUTHTOK_EXPIRED          18     /* Password expired and no longer */
#define  PAM_SESSION_ERR              19     /* can not make/remove entry for */
                                             /* specified session */
#define  PAM_AUTHTOK_ERR              20     /* Authentication token */
                                             /*   manipulation error */
#define  PAM_AUTHTOK_RECOVERY_ERR     21     /* Old authentication token */
                                      /*   cannot be recovered */
#define  PAM_AUTHTOK_LOCK_BUSY        22     /* Authentication token */
                                             /*   lock busy */
#define  PAM_AUTHTOK_DISABLE_AGING    23     /* Authentication token aging */
                                             /*   is disabled */
#define  PAM_NO_MODULE_DATA           24     /* module data not found */
#define  PAM_IGNORE                   25     /* ignore module */
#define  PAM_ABORT                    26     /* General PAM failure */
#define  PAM_TRY_AGAIN                27     /* Unable to update password */
                                             /* Try again another time */
#define  PAM_MODULE_UNKNOWN           28     /* Module unknown */
```

```
#define  PAM_DOMAIN_UNKNOWN            29    /* Domain unknown */

/*
 * structure pam_message is used to pass prompt, error message,
 * or any text information from scheme to application/user.
 */

struct pam_message {
      int msg_style;            /* Msg_style - see below */
      char *msg;                /* Message string */
};

/*
 * msg_style defines the interaction style between the
 * scheme and the application.
 */
#define  PAM_PROMPT_ECHO_OFF     1      /* Echo off when getting response */
#define  PAM_PROMPT_ECHO_ON      2      /* Echo on when getting response */
#define  PAM_ERROR_MSG           3      /* Error message */
#define  PAM_TEXT_INFO           4      /* Textual information */

/*
 * max # of messages passed to the application through the
 * conversation function call
 */
#define  PAM_MAX_NUM_MSG   32

/*
 * max size (in chars) of each messages passed to the application
 * through the conversation function call
 */
#define  PAM_MAX_MSG_SIZE  512

/*
 * max size (in chars) of each response passed from the application
 * through the conversation function call
 */
#define  PAM_MAX_RESP_SIZE 512

/*
 * structure pam_response is used by the scheme to get the user's
 * response back from the application/user.
 */

struct pam_response {
```

```
        char *resp;                /* Response string */
        int resp_retcode;          /* Return code - for future use */
};

/*
 * structure pam_conv is used by authentication applications for passing
 * call back function pointers and application data pointers to the scheme
 */
struct pam_conv {
        int (*conv)(int, struct pam_message **,
            struct pam_response **, void *);
        void *appdata_ptr;               /* Application data ptr */
};

/* the pam handle */
typedef struct pam_handle pam_handle_t;

/*
 * pam_start() is called to initiate an authentication exchange
 * with PAM.
 */
extern int
pam_start(
        const char *service_name,        /* Service Name */
        const char *user,                /* User Name */
        const struct pam_conv *pam_conv, /* Conversation structure */
        pam_handle_t **pamh              /* Address to store handle */
);

/*
 * pam_end() is called to end an authentication exchange with PAM.
 */
extern int
pam_end(
        pam_handle_t *pamh,        /* handle from pam_start() */
        int status                 /* the final status value that */
                                   /* gets passed to cleanup functions */
);

/*
 * pam_set_item is called to store an object in PAM handle.
 */
extern int
pam_set_item(
        pam_handle_t *pamh,        /* PAM handle */
```

```
      int item_type,            /* Type of object - see below */
      const void *item          /* Address of place to put pointer */
                                /*   to object */
);

/*
 * pam_get_item is called to retrieve an object from the static data area
 */
extern int
pam_get_item(
      const pam_handle_t *pamh, /* PAM handle */
      int item_type,            /* Type of object - see below */
      void **      item         /* Address of place to put pointer */
                                /*   to object */
);

/* Items supported by pam_[sg]et_item() calls */
#define  PAM_SERVICE            1      /* The program/service name */
#define  PAM_USER         2      /* The user name */
#define  PAM_TTY          3      /* The tty name */
#define  PAM_RHOST        4      /* The remote host name */
#define  PAM_CONV         5      /* The conversation structure */
#define  PAM_AUTHTOK            6      /* The authentication token */
#define  PAM_OLDAUTHTOK   7      /* Old authentication token */
#define  PAM_RUSER        8      /* The remote user name */
#define  PAM_USER_PROMPT  9      /* The user prompt */

/*
 * pam_get_user is called to retrieve the user name (PAM_USER). If PAM_USER
 * is not set then this call will prompt for the user name using the
 * conversation function. This function should only be used by modules, not
 * applications.
 */

extern int
pam_get_user(
      pam_handle_t *pamh,       /* PAM handle */
      char **user,              /* User Name */
      const char *prompt        /* Prompt */
);

/*
 * pam_set_data is used to create module specific data, and
 * to optionally add a cleanup handler that gets called by pam_end.
 *
```

```
 */
extern int
pam_set_data(
       pam_handle_t *pamh,               /* PAM handle */
       const char *module_data_name,     /* unique module data name */
       const void *data,                 /* the module specific data */
       void (*cleanup)(pam_handle_t *pamh, void *data, int pam_end_status)
);

/*
 * get module specific data set by pam_set_data.
 * returns PAM_NO_MODULE_DATA if specified module data was not found.
 */
extern int
pam_get_data(
       const pam_handle_t *pamh,
       const char *module_data_name,
       void **data
);

/*
 * PAM equivalent to strerror();
 */
extern char *
pam_strerror(
       pam_handle_t *pamh, /* pam handle */
       int errnum          /* error number */
);

/* general flag for pam_* functions */
#define  PAM_SILENT 0x80000000

/*
 * pam_authenticate is called to authenticate the current user.
 */
extern int
pam_authenticate(
       pam_handle_t *pamh,
       int flags
);

/*
 * Flags for pam_authenticate
 */
```

```
#define  PAM_DISALLOW_NULL_AUTHTOK 0x1  /* The password must be non-null */

/*
 * pam_authenticate_secondary is called to authenticate the current user
to a secondary domain.
 */
extern int
pam_authenticate_secondary (
       pam_handle_t * pamh,
       char * target_username,
       char * target_module_type,
       char * target_authn_domain,
       char * target_supp_data,
       unsigned char * target_module_authtok,
       int flags
);

/*
 * pam_acct_mgmt is called to perform account management processing
 */
extern int
pam_acct_mgmt(
       pam_handle_t *pamh,
       int flags
);

/*
 * pam_open_session is called to note the initiation of new session in the
 * appropriate administrative data bases.
 */
extern int
pam_open_session(
       pam_handle_t *pamh,
       int flags
);

/*
 * pam_close_session records the termination of a session.
 */
extern int
pam_close_session(
       pam_handle_t *pamh,
       int          flags
);
```

```
/* pam_setcred is called to set the credentials of the current user */
extern int
pam_setcred(
       pam_handle_t *pamh,
       int flags
);

/* flags for pam_setcred() */
#define  PAM_ESTABLISH_CRED      0x1    /* set scheme specific user id */
#define  PAM_DELETE_CRED         0x2    /* unset scheme specific user id */
#define  PAM_REINITIALIZE_CRED   0x4    /* reinitialize user credentials */
                                        /* (after a password has changed */
#define  PAM_REFRESH_CRED        0x8    /* extend lifetime of credentials */

/* pam_chauthtok is called to change authentication token */

extern int
pam_chauthtok(
       pam_handle_t  *pamh,
       int           flags
);

/*
 * Be careful - there are flags defined for pam_sm_chauthtok() in
 * pam_modules.h also.
 */
#define  PAM_CHANGE_EXPIRED_AUTHTOK     0x4 /* update expired passwords only */

/* pam_getenv is called to retrieve the value of a PAM environemnt variable     */

char* pam_getenv (
       pam_handle_t * pamh,
       const char * name,
);

/* pam_getenvlist is called to retrieve a list of all the PAM environemnt variables  */

char** pam_getenvlist (
       pam_handle_t * pamh,
);

/* pam_putenv sets the value of a PAM environment variable */

int pam_putenv (
       pam_handle_t * pamh,
```

```
        const char * namevalue,
);

/*  pam_get_mapped_authok gets a mapped password for the user  */

int pam_get_mapped_authtok (
        pam_handle_t * pamh,
        const char * target_module_username,
        const char * target_module_type,
        const char * target_authn_domain,
        size_t * target_authtok_len,
        unsigned char ** target_module_authtok
);

/* pam_set_mapped_authtok stores a mapped password for the username supplied */

int pam_set_mapped_authtok (
        pam_handle_t * pamh,
        char * target_module_username,
        size_t * target_authtok_len,
        unsigned char * target_module_authtok,
        char * target_module_type,
        char * target_authn_domain,
);

/* pam_get_mapped_username retrieves a valid matched identity in a new domain  */

int pam_get_mapped_username (
        pam_handle_t * pamh,
        const char * src_username,
        const char * src_module_type,
        const char * src_authn_domain,
        const char * target_module_type,
        const char * target_authn_domain,
        char ** target_module_username,
);

/* pam_set_mapped_username stores a mapped name for the username supplied */

int pam_set_mapped_username (
        pam_handle_t * pamh,
        char * src_username,
        char * src_module_type,
        char * src_authn_domain,
        char * target_module_username,
```

```
        char * target_module_type,
        char * target_authn_domain,
);

/* pam_get_user retrieves the current user name from a PAM handle */

int pam_get_user (
        pam_handle_t * pamh,
        char ** user,
        const char * prompt
);
```

A.2 PAM_MODULE.H

```
extern int
pam_sm_authenticate(
        pam_handle_t *pamh,
        int    flags,
        int    argc,
        const char   **argv);

extern int
pam_sm_authenticate_secondary(
        pam_handle_t *pamh,
        int    flags,
        int    argc,
        const char   **argv);

extern int
pam_sm_setcred(
        pam_handle_t *pamh,
        int    flags,
        int    argc,
        const char   **argv);

extern int
pam_sm_acct_mgmt(
        pam_handle_t *pamh,
        int    flags,
        int    argc,
        const char   **argv);

extern int
pam_sm_open_session(
        pam_handle_t *pamh,
        int    flags,
        int    argc,
        const char   **argv);

extern int
pam_sm_close_session(
        pam_handle_t *pamh,
        int    flags,
        int    argc,
        const char   **argv);

/*
 * Be careful - there are flags defined for pam_chauthtok() in
```

```
 * pam_appl.h also.
 */
#define  PAM_PRELIM_CHECK  0x1
#define  PAM_UPDATE_AUTHTOK      0x2

extern int
pam_sm_chauthtok(
      pam_handle_t *pamh,
      int          flags,
      int          argc,
      const char   **argv);

extern int
pam_sm_get_mapped_authtok(
      pam_handle_t *pamh,
      int    flags,
      int    argc,
      const char   **argv);

extern int
pam_sm_set_mapped_authtok(
      pam_handle_t *pamh,
      int    flags,
      int    argc,
      const char   **argv);

extern int
pam_sm_get_mapped_username(
      pam_handle_t *pamh,
      int    flags,
      int    argc,
      const char   **argv);

extern int
pam_sm_set_mapped_username(
      pam_handle_t *pamh,
      int    flags,
      int    argc,
      const char   **argv);
```

Appendix B

PAM Configuration Administration

The syntax and semantics of the PAM configuration data are part of this specification. However, the physical form in which that configuration data is stored is not specified.

Existing implementations of PAM utilise a file, pam.conf, located on each platform supporting PAM to hold the PAM configuration data. It is not a requirement of this specification that an implementation needs to support a local pam.conf file. A centralised service could be provided for ease of administration.

The current PAM configuration syntax supports the definition of PAM behaviour on a host/service basis. The possibility of extending the PAM configuration file to include user identities and thus support a host/user/service basis has been discussed during the development of this specification but has been deferred to a future version.

This specification does not include any definition of the way in which underlying modules are configured, except where module specific parameters are to be included in the options field as part of an entry within the PAM configuration data. The purpose and value of this specification is to define a common interface to authentication and other types of modules. The mechanism by which the modules used are configured is implementation specific.

B.1 Mapping Service Configuration

The mapping service is configured through the PAM configuration file. Such services are identified by a PAM type *mapping*. Just like the other PAM types, *mapping* modules can be specified on a per service basis or for all services through the "OTHER" entry. Multiple mapping modules can also be stacked.

It is not a requirement for the module providers to use mapped user names or passwords. If mapping is supported, the module provider should document whether mapping is provided and whether for user names, passwords, or both. The system administrator can enable mapping on a per application basis through the use of option flags such as "*map=user*" or "*map=pass*". Based on such flags, the modules should call *pam_get_user*( ) or *pam_get_mapped_username*( ) to get the name of the user.

It is the responsibility of the PAM engine to load the mapping services and route the "get" and "set" calls to the appropriate mapping services, and return appropriate status to the calling module. This is very similar to the way PAM engine handles other PAM types such as *auth*.

If there are multiple mapping modules, the semantics of the control flags (i.e., *optional*, *required*, and *sufficient*) require special attention. With multiple mapping services, the desired semantics are that the result from the first successful map will be used. For updates, the new map should be sent to all the configured mapping services so that the appropriate mapping service gets updated. Thus for the *get* calls, the semantics required are of the *sufficient* flag, and for the *set* calls, the semantics required are of the *optional* flag.

B.2 Module Option Parameters

Modules may be supplied with parameters via the options field of a PAM configuration entry. The recommended syntax is:

```
   option
or option_name=option_value
```

WHITE SPACE is delimiter between options

Proprietary option names may be used. Collision with "standard" names is considered to be unlikely and easily addressed by one or other names being changed.

Option names should be registered with the OpenGroup. The list of registered names is published by the OpenGroup at:

```
http://????????
```

That document also describes the procedure for registering new option names.

B.3 Additional PAM Options

The introduction of options in the PAM configuration are necessary to accommodate mapping and additional authentication attempts. The following options are defined:

map=user
The module should do mapping for user names.

map=pass
The module should do mapping for passwords.

map
The module should do mapping for user names and passwords.

retry_user
If an authentication service retrieves an invalid user name (this could be due to unsynchronised database), the authentication would fail. If an option such as *retry_user* were present in the configuration file, the authentication service should prompt the user for another name and reattempt authentication. If this user name is valid, it should update the mapping database, using the appropriate PAM *set* mapping routines.

retry_pass
If an authentication service retrieves an invalid password (this could be due to unsynchronised database), the authentication would fail. If an option such as *retry_pass* were present in the configuration file, the authentication service should prompt the user for another password and reattempt authentication. If this password is valid, it should update the mapping database, using the appropriate PAM *set* mapping routines.

Appendix C

Internationalisation

The issue of codesets is viewed by the working group as a general issue that requires addressing globally. It would be to parochial to try and address currently in PAM.

This appendix documents the issue and some proposals as presented to and discussed by the working group.

C.1 Introduction

The PAM architecture assumes that usernames and passwords are selected from the same codeset and handled using C programming character pointers. This may not be true since there are a variety of codesets available. In addition, not all characters sets are handled using C character pointers. A proposal from Hewlett-Packard recommends a parameter to maintain the codeset type.

C.2 Single System Codesets

A username and corresponding password are compared against stored data, (/etc/passwd on a simple UNIX machine). On a stand-alone machine, the comparison data and the interface which captures data, utilise the same codeset.

A simple byte by byte match between the text collected by the interface and the comparison data may be used for the username. Yet, passwords comparisons are not so simple since it is dependent upon the codeset by which the password is represented.

For example, let us consider a user with password "ABCD" on a system where the password comparison data is formed in the UNIX manner.

If the system uses the ASCII codeset, an encryption key is generated from {0x41, 0x42, 0x43, 0x44} and a fixed plaintext is encrypted with this key. If the system uses EBCDIC, the encryption key is generated from {0xC1, 0xC2, 0xC3, 0xC4}.

Therefore, even if the plaintext is encrypted using identical algorithms on these two systems, the results will differ due to the codeset.

C.3 Usernames

If the login interface and the authentication service (AS) do not reside on the same machine, it is not reasonable to assume compatible codesets are deployed.

For example, in a Japanese enterprise where the username is in Kanji, a user logs into a machine which uses Shift-JIS. But suppose the authentication service is based upon eucJP. In order to achieve successful authentication, the Shift-JIS username must be recognised by the eucJP service.

C.4 Passwords

A system may restrict passwords to a particular set of characters which exist in all versions of EBCDIC and localisations of ISO 646, such as ASCII. The system may also perform conversions to ISO 646 prior to processing. Although this guarantees consistent comparison data, it may be undesirable. Technically, this approach is not sufficient - as the character set decreases, passwords become weaker. In addition, it is inconvenient to use an unfamiliar set of characters for password selection.

C.5 Proposed Solution

From the previous example, a user has a Kanji password (Shift-JIS codeset) but an AS which requires a different codeset, eudJP. One potential solution involves passing the Shift-JIS password, with a tag indicating the codeset type, from the client to the AS though a protected channel. The AS converts the password to eucJP and proceeds with its comparison to the stored data.

The stored value in the remote AS differs from the encrypted Shift-JIS password (on the local machine) since different codesets are employed. If local authentication is required, password synchronisation between the machines may not be achieved by simply copying the remote AS's stored value to the client, even though both use the same encryption algorithm.

Further complications arise if the client attempts to convert the password to the codeset used by the remote AS. That is, the client must determine the codeset used by the AS. In addition, if local authentication is required, the client must not only verify the type of codeset applied to generate the data, but also securely retrieve the stored value from the remote AS.

Regardless of codesets, authentication services must function properly.

C.6 Smart Cards

The issues discussed above apply to smart cards which are used primarily as a repository of passwords and computations are performed elsewhere. If passwords on a card are indexed by username, there may be complications if identical usernames are used on machines with different codesets. However, if passwords are indexed by system name and username pairs, smart cards may hold passwords in the localised codeset.

Appendix D

XSSO Account Management Services

The essential objective of XSSO Account Management services is to provide support for the development of applications to implement co-ordinated management of account information bases in a heterogeneous set of security domains (platforms and applications).

As described in Chapter 1, an individual end-user of a distributed system typically requires the capability to utilise many different platforms and applications (domains). To provide this capability an individual needs to be represented by entries in the account information bases appropriate to each of those domains.

Each individual domain will have its own set of account attributes to be managed. These are referenced to domain specific identities and have domain specific formats. The requirement on XSSO is to provide the capability of managing a minimum common core set of such attributes with the capability for future extension.

Note: The business benefit sought from the specification of the XSSO ACM Service is a reduction in the number of administrative interactions with the system to manage individual accounts. In an approach based on a minimum common core set of attributes, the extent to which these benefits are realisable within a particular system depend upon an assumption that any additional attributes required for accounts by a particular domain can be generated on basis of domain defined defaults or derived from XSSO provided attribute values on the basis of domain defined rules. As an example, under DCE the creation of a login account requires the previous creation of the appropriate principal, group and organisation registry entries.

D.1 Scope of XSSO Account Management

D.1.1 XBSS Functional Requirements

The core requirements of XSSO Account Management are based upon requirements detailed in the XBSS. These encompass both Account Level Policies and System Level Policies.

Account Level Policy

The following requirements are applicable to the administration of individual accounts:

- by default all accounts within the scope of a policy domain shall be uniquely identified for the purposes of authentication and security audit. This may be achieved by the use of a single unique identity within the policy domain or by a one to one mapping of primary domain identity to secondary domain identity if a single identity representation cannot be used for all services within the policy domain.

  Note: It is acceptable for identities or attributes used solely for the purposes of authorisation to be shared by several principals. Examples of such use are for the support of roles and anonymous ftp services.

- authentication data shall include information for verifying the identity of individual principals

- policy attributes of individual principals shall be maintained (for example, groups, time intervals, location.) At least a primary group is mandatory

Note: Some implementations support the concept of supplementary groups in which a principal may exercise the rights applicable to a set of groups concurrently.

Within the XSSO, the concept of a group is subsumed as an account attribute and is not separately distinguished.

- an administrator shall be able to enable and disable an account
- an administrator shall be able to enquire of status information for all active principals and all accounts (enabled or disabled)
- initialise and change an account password
- set password expiration at any time, including before first use
- audit event pre-selection mask by account.

System Level Policy

The following XBSS requirements are applicable and controlled on a policy domain wide basis:

- an administrator shall be able to configure the period by which subsequent sign-on attempts are delayed after a failed sign-on attempt
- configure password ageing parameters
- configure period and frequency of end-user notification of password expiry
- configure password re-use controls on the basis of at least one of:
  - period of time
  - number of password changes
  - minimum change period
- configure required complexity of end-user entered passwords.

D.1.2 Basic Functional Requirements

The XSSO ACM-API is required to support the following functions within the domain supporting the interface:

Add Account

Create a new account.

Delete Account

Delete account.

Note: If done thoroughly, this encompasses the deletion or reassignment of system resources currently assigned to the account according to the applicable policy. For example, files owned by the account should be deleted or have their ownership changed. Access authorisations specifically assigned for the account should be removed. For example, this may imply a change of ACLs and file ownership attributes.

Update Account

Change the value of account attributes.

List Accounts

List all accounts or accounts based on optional selection criteria. This may include the selective listing of "active accounts" for those domains in which the interpretation of an "active account" is defined.

Disable/Enable Account
Disable and enable the use of an account to access the services of the domain. The disabling of an account should include, where possible, the disabling of any scheduled sessions for the account and the termination of any interactive sessions for the account that are in progress at the time the account is disabled.

Update Account Authentication Information
This includes update of account authentication information by an administrator, including a forced change on first usage, and by the end-user via XSSO Sign-on Services. An administrator may also be able to update the method or methods of authentication. In the first version of XSSO the only form of authentication information required to be supported are passwords and pass phrases but the specification should not preclude future extensibility.

Add Account Attribute
Create a new attribute that may be assigned to accounts.

Delete Account Attribute
Delete an attribute. Where possible this should include the removal of this attribute from all accounts, or should only succeed if the attribute is not currently assigned to any account.

Set Account Attributes
Set the values of attributes for an account.

Get Account Attributes
Retrieve the attributes assigned to an account.

D.2 Account Management Authorities

The account management authorities identified below are disjoint authorities that are used to control the exercise of Account Management Functions in support of the principles of Least privilege and Separation of Duties.

XSSO_ACCOUNT_ADMIN
Required to create, populate, and modify accounts. Excludes modification of audit attributes, authentication information and account enabling, disabling and deletion.

XSSO_ATTRIBUTE_ADMIN
Required to assign attributes to accounts. This authority may be parameterised to control the modification of discrete sets of attributes.

Note: The next two authorities can be considered as specific examples of the XSSO_ATTRIBUTE_ADMIN authority with the sets of attributes which they cover having particular security significance.

XSSO_ACCOUNT_AUDITOR
Required to modify account audit attributes.

XSSO_ACCOUNT_SECURITY
Required to set and modify account authentication information attributes, and to enable, disable, and delete accounts.

XSSO_ADMIN
Required to configure XSSO UAM services. For example to add/modify/delete *domain types* and *domain instances*.

D.3 Common Core Account Attributes

The common core attributes for accounts represent the minimum set of attributes required by XBSS for the purposes of both account level and system level policy enforcement. These attributes are not necessarily applicable nor mappable to all component account registries within a policy domain but a XSSO implementation has to be capable of supporting their management. The common core attributes comprise:

Account_name
Human readable text string. Must be unique within the domain of definition.

Account_ID
Domain internal accountID that must be unique within the domain of definition.

Account Enabled Flag
Account enabled if flag set, account disabled if unset.

Account Last Used
Date and time the account was last used in an authentication operation.

Authentication Information
For the current specification this comprises at least a password. In the future it may include information such as authentication method.

Authentication Information Expiry
The date and time at which the current authentication information expires at which system level policy may enforce an authentication information change. Appropriate setting of this attribute can force an authentication information change on the next use of the account for a authentication operation.

Account Audit Identity
Identity assigned to an account for audit purposes within domain. (May be the same as Account_ID.)

Account Audit Pre-selection Mask
Configuration of audit profile for account for those component services that support the control of audit service by account.

Authorisation Attributes
Set of authorisations assigned to account. These may include access identities, roles identities, service specific privileges, etc. These may be specific to each component service within a XSSO policy domain.

Account Environment Data
Information that defines the environment to be created on session establishment within which the account is required to operate. For example, shell or application, home directory, environment variables, resource quotas, default resource access permissions, etc. These will be specific to each component service within a XSSO policy domain.

The interface style required to support XSSO is that in which the minimum information necessary to identify the account is always required and all parameters, whether core or extended are passed as a list of attribute/value pairs as and when required.

Note: The issue of atomicity of operations and locking has been discussed and the need to identify any implicit locking mechanisms in the use of the interface identified. This is only of concern in update operations. The semantics of the operation require defining. There are two possible approaches:

- Replace complete account information with information supplied in function call, or
- Add or replace only the specific information included in the function call.

D.4 Management of Account Information for Multiple Services

A concept of XSSO Account Administration is that XSSO provides a common management interface to account information for all the component services within an XSSO Policy Domain.

Each component service may identify its account attributes by different labels and represent them in different formats. Each domain may also utilise different naming schemes for account identities. It is therefore necessary that functionality is included in an implementation of XSSO UAM to enable an administrator to configure the mapping and XSSO sign-on components, such as PAM, to be able to get the mapping to support secondary sign-on operations.

Such mappings may be based on two approaches:

- **Algorithmic**
  In which the secondary domain representation of an identity or attribute may be derived from the primary domain representation on the basis of a transformation rule
- **Administrative**
  In which the mapping cannot be derived on the basis of a transformation rule but has to be explicitly defined.

The XSSO administrative application management information base requires to hold the following information:

Register of Domain Types
This is a register of the different types of domains that are integrated with and managed by the XSSO administrative application. For each domain type, a list of the domain attributes is maintained.

Register of Domain Instances
This is a register of instances of domains managed by the XSSO administrative application. For each instance of a domain, its location and domain type is maintained.

Register of Accounts
This is a register of the accounts managed by the XSSO administrative application. For each account a list of the domain instances within which it is registered and the attributes assigned to it within each domain instance are maintained. This may include copies of the credentials required to authenticate to the domain if required to support secondary sign-on.

The XSSO may not necessarily maintain a copy of the attributes assigned within each domain instance provided that it can retrieve the information from the domain itself when required.

Default Account Profiles
To facilitate the assignment of attributes across multiple domains to accounts and the simultaneous registration of principals in multiple domains then a set of account profiles may be defined. These may include rules for derivation or look up of secondary domain specific attributes based on XSSO domain based attributes.

D.4.1 Registry of Domain Types

For each domain typ, a set of attributes is required to be registered. For each attribute the following information may be required:

Domain_Type
The domain type to which the attribute belongs.

Attribute_ID
The internal label by which the attribute is referenced.

Attribute_Name
The human readable label used to reference the attribute.

Attribute_Format (Encoding)
The format in which the attribute value is represented for storage and exchange.

Attribute_Default_Value
The rule by which the default value of this attribute is derived.

Domain_Authorities
The domain specific authorities that have to be possessed by a caller of the ACM-API in order to query or modify this attribute.

D.5 XSSO Account Management Implementation Considerations

D.5.1 Mapping of Administrative Authorities to XSSO UAM Agents

An XSSO implementation needs to be able to map the domain specific authorities required to manage the attributes within the domain to the XSSO administrative authorities and roles that it supports. That is, an XSSO agent requires sufficient privilege to manage the domain that it services. The exercise of those privileges must be matched to the authorisation assigned to the initiating administrator principals.

The security context of an XSSO agent invoked within a specific domain on behalf of an XSSO Administrator is required to reflect the security context of the XSSO Administrator.

D.5.2 XSSO Management Information Base Initialisation

An implementation is required to define how the XSSO Management Information Base is initialised.

Glossary

access control
The prevention of unauthorised use of a resource including the prevention of use of a resource in an unauthorised manner (see ISO/IEC 7498-2).

access control information
(ACI) — any information used for access control purposes, including contextual information (see ISO/IEC 10081-3).

access control policy
The set of rules that define the conditions under which an access may take place (see ISO/IEC 10081-3).

accountability
The property that ensures that the actions of an entity may be traced to that entity (see ISO/IEC 7498-2).

ACI
Access control information.

ACL
Access control list.

action
The operations and operands that form part of an attempted access (see ISO/IEC 10081-3).

active threat
The threat of a deliberate unauthorised change to the state of the system

administrative security information
Persistent information associated with entities; it is conceptually stored in the Security Management Information Base. Examples are:

- security attributes associated with users and set up on user account installation, which is used to configure the user's identity and privileges within the system
- information configuring a secure interaction policy between one entity and another entity, which is used as the basis for the establishment of operational associations between those two entities.

API
Application Programming Interface.

The interface between the application software and the application platform, across which all services are provided.

The application programming interface is primarily in support of application portability, but system and application interoperability are also supported by a communication API (see POSIX.0).

assertion
Explicit statement in a system security policy that security measures in one security domain constitute an adequate basis for security measures (or lack of them) in another (see CESG Memo).

audit
See Security Audit (see ISO/IEC 7498-2).

audit authority
The manager responsible for defining those aspects of a security policy applicable to maintaining a security audit (see ISO/IEC 10081-7).

audit trail
See Security Audit Trail (see ISO/IEC 7498-2).

authenticated identity
An identity of a principal that has been assured through authentication (see ISO/IEC 10081-2).

authentication
Verify claimed identity; see data origin authentication, and peer entity authentication (see ISO/IEC 7498-2).

authentication certificate
Authentication information in the form of a security certificate which may be used to assure the identity of an entity guaranteed by an authentication authority (see ISO/IEC 10081-2).

authentication exchange
A sequence of one or more transfers of exchange authentication information (AI) for the purposes of performing an authentication (see ISO/IEC 10081-2).

authentication information (AI)
Information used to establish the validity of a claimed identity (see ISO/IEC 7498-2).

authentication initiator
The entity which starts an authentication exchange (see ISO/IEC 10081-2).

authentication method
Method for demonstrating knowledge of a secret. The quality of the authentication method, its strength is determined by the cryptographic basis of the key distribution service on which it is based. A symmetric key based method, in which both entities share common authentication information, is considered to be a weaker method than an asymmetric key based method, in which not all the authentication information is shared by both entities.

authorisation
The granting of rights, which includes the granting of access based on access rights (see ISO/IEC 7498-2).

authorisation policy
A set of rules, part of an access control policy, by which access by security subjects to security objects is granted or denied. An authorisation policy may be defined in terms of access control lists, capabilities or attributes assigned to security subjects, security objects or both (see ECMA TR/46).

availability
The property of being accessible and usable upon demand by an authorised entity (see ISO/IEC 7498-2).

claim authentication information
(Claim AI) — information used by a claimant to generate exchange AI needed to authenticate a principal (see ISO/IEC 10081-2).

clear text
Intelligible data, the semantic content of which is available (see ISO/IEC 7498-2).

client-server
These operations occur between a pair of communicating independent peer processes. The peer process initiating a service request is termed the client. The peer process responding to a service request is termed the server. A process may act as both client and server in the context of a set of transactions.

confidentiality
The property that information is not made available or disclosed to unauthorised individuals, entities, or processes (see ISO/IEC 7498-2).

contextual information
Information derived from the context in which an access is made (for example, time of day) (see ISO/IEC 10081-3).

corporate security policy
The set of laws, rules and practices that regulate how assets including sensitive information are managed, protected and distributed within a user organisation (see ITSEC).

countermeasure
The deployment of a set of security services to protect against a security threat.

credentials
Data that is transferred to establish the claimed identity of an entity (see ISO/IEC 7498-2).

data integrity
The property that data has not been altered or destroyed in an unauthorised manner (see ISO/IEC 7498-2).

data origin authentication
The corroboration that the entity responsible for the creation of a set of data is the one claimed.

denial of service
The unauthorised prevention of authorised access to resources or the delaying of time-critical operations (see ISO/IEC 7498-2).

digital fingerprint
A characteristic of a data item, such as a cryptographic checkvalue or the result of performing a one-way hash function on the data, that is sufficiently peculiar to the data item that it is computationally infeasible to find another data item that possesses the same characteristics (see ISO/IEC 10081-1).

digital signature
Data appended to, or a cryptographic transformation (see cryptography) of, a data unit that allows a recipient of the data unit to prove the source and integrity of the data unit and protect against forgery for example, by the recipient (see ISO/IEC 7498-2).

discretionary access control
A discretionary authorisation scheme is one under which any principal using the domain services may be authorised to assign or modify ACI such that he may modify the authorisations of other principals under the scheme. A typical example is an ACL scheme which is often referred to as Discretionary Access Control (DAC).

distinguishing identifier
Data that unambiguously distinguishes an entity in the authentication process. Such an identifier shall be unambiguous at least within a security domain (see ISO/IEC 10081-2).

distributed application
A set of information processing resources distributed over one or more open systems which provides a well-defined set of functionality to (human) users, to assist a given (office) task (see ECMA TR/46).

exchange authentication information
(Exchange AI) — information exchanged between a claimant and a verifier during the process of authenticating a principal (see ISO/IEC 10081-2).

identification
The assignment of a name by which an entity can be referenced. The entity may be high level (such as a user) or low level (such as a process or communication channel.

identity-based security policy
A security policy based on the identities or attributes of users, a group of users, or entities acting on behalf of the users and the resources or targets being accessed (see ISO/IEC 7498-2).

initiator
An entity (for example, human user or computer based entity) that attempts to access other entities (see ISO/IEC 10081-3).

integrity
See Data Integrity (see ISO/IEC 7498-2).

masquerade
The unauthorised pretence by an entity to be a different entity (see ISO/IEC 7498-2).

non-discretionary access control
A non-discretionary authorisation scheme is one under which only the recognised security authority of the security domain may assign or modify the ACI for the authorisation scheme such that the authorisations of principals under the scheme are modified.

off-line authentication certificate
A particular form of authentication information binding an entity to a cryptographic key, certified by a trusted authority, which may be used for authentication without directly interacting with the authority (see ISO/IEC 10081-2).

on-line authentication certificate
A particular form of authentication information, certified by a trusted authority, which may be used for authentication following direct interaction with the authority (see ISO/IEC 10081-2).

operational security information
Transient information related to a single operation or set of operations within the context of an operational association, for example, a user session. Operational security information represents the current security context of the operations and may be passed as parameters to the operational primitives or retrieved from the operations environment as defaults.

organisational security policy
Set of laws, rules, and practices that regulates how an organisation manages, protects, and distributes sensitive information (see Federal Criteria).

password
Confidential authentication information, usually composed of a string of characters (see ISO/IEC 7498-2).

peer-entity authentication
The corroboration that a peer entity in an association is the one claimed (see ISO/IEC 7498-2).

physical security
The measures used to provide physical protection of resources against deliberate and accidental threats (see ISO/IEC 7498-2).

platform domain
A security domain encompassing the operating system, the entities and operations it supports and its security policy.

policy
See security policy (see ISO/IEC 7498-2).

primary service
An independent category of service such as operating system services, communication services and data management services. Each primary service provides a discrete set of functionality. Each primary service inherently includes generic qualities such as usability, manageability and security.

Security services are therefore not primary services but are invoked as part of the provision of primary services by the primary service provider.

principal
An entity whose identity can be authenticated (see ISO/IEC 10081-2).

privacy
The right of individuals to control or influence what information related to them may be collected and stored and by whom and to whom that information may be disclosed.

Note: because this term relates to the right of individuals, it cannot be very precise and its use should be avoided except as a motivation for requiring security (see ISO/IEC 7498-2).

quality of protection
A label that implies methods of security protection under a security policy. This normally includes a combination of integrity and confidentiality requirements and is typically implemented in a communications environment by a combination of cryptographic mechanisms.

repudiation
Denial by one of the entities involved in a communication of having participated in all or part of the communication (see ISO/IEC 7498-2).

rule-based security policy
A security policy based on global rules imposed for all users. These rules usually rely on a comparison of the sensitivity of the resources being accessed and the possession of corresponding attributes of users, a group of users, or entities acting on behalf of users (see ISO/IEC 7498-2).

seal
A cryptographic checkvalue that supports integrity but does not protect against forgery by the recipient (that is, it does not support non-repudiation). When a seal is associated with a data element, that data element is *sealed* (see ISO/IEC 10081-1).

secondary discretionary disclosure
An example of the misuse of access rights. It occurs when a principal authorised to access some information copies that information and authorises access to the copy by a second

principal who is not authorised to access the original information.

secret key
In a symmetric cryptographic algorithm the key shared between two entities (see ISO/IEC 10081-1).

secure association
An instance of secure communication (using communication in the broad sense of space and/or time) which makes use of a secure context.

secure context
The existence of the necessary information for the correct operation of the security mechanisms at the appropriate place and time.

secure interaction policy
The common aspects of the security policies in effect at each of the communicating application processes (see CESG Memo).

security architecture
A high level description of the structure of a system, with security functions assigned to components within this structure (see CESG Memo).

security attribute
A security attribute is a piece of security information which is associated with an entity.

security audit
An independent review and examination of system records and operations in order to test for adequacy of system controls, to ensure compliance with established policy and operational procedures, to detect breaches in security and to recommend any indicated changes in control, policy and procedures (see ISO/IEC 7498-2).

security audit trail
Data collected and potentially used to facilitate a security audit (see ISO/IEC 7498-2).

security auditor
An individual or a process allowed to have access to the security audit trail and to build audit reports (see ISO/IEC 10081-7).

security aware
The caller of an API that is aware of the security functionality and parameters which may be provided by an API.

security certificate
A set of security-relevant data from an issuing security authority that is protected by integrity and data origin authentication, and includes an indication of a time period of validity (see ISO/IEC 10081-1).

Note: All certificates are deemed to be security certificates (see the relevant definitions in ISO/IEC 7498-2) adopted in order to avoid terminology conflicts with (that is the directory authentication standard).

security domain
A set of elements, a security policy, a security authority and a set of security-relevant operations in which the set of elements are subject to the security policy, administered by the security authority, for the specified operations (see ISO/IEC 10081-1).

security event manager
An individual or process allowed to specify and manage the events which may generate a security message and to establish the action or actions to be taken for each security message

type (see ISO/IEC 10081-7).

security label
The marking bound to a resource (which may be a data unit) that names or designates the security attributes of that resource (see ISO/IEC 7498-2).

Note: The marking may be explicit or implicit.

security policy
The set of criteria for the provision of security services (see also identity-based and rule-based security policy).

security service
A service which may be invoked directly or indirectly by functions within a system that ensures adequate security of the system or of data transfers between components of the system or with other systems.

security state
State information that is held in an open system and which is required for the provision of security services.

security token
A set of security-relevant data that is protected by integrity and data origin authentication from a source that is not considered a security authority (see ISO/IEC 10081-1).

security unaware
The caller of an API that is unaware of the security functionality and parameters which may be provided by an API.

service domain
A security domain encompassing an application, the entities and operations it supports and its security policy.

signature
See digital signature (see ISO/IEC 7498-2).

strength of mechanism
An aspect of the assessment of the effectiveness of a security mechanism, namely the ability of the security mechanism to withstand direct attack against deficiencies in its underlying algorithms, principles and properties (see ITSEC).

system security function
A capability of an open system to perform security-related processing (see CESG Memo).

target
An entity to which access may be attempted (see ISO/IEC 10081-3).

threat
A potential violation of security (see ISO/IEC 7498-2).
An action or event that might prejudice security (see ITSEC).

trap door
A hidden software or hardware mechanism that permits system protection mechanisms to be circumvented. It is activated in some non-apparent manner (for example, special ''random'' key sequence at a terminal) (see TCSEC).

trojan horse
Computer program containing an apparent or actual useful function that contains additional (hidden) functions that allow unauthorised collection, falsification or destruction of data (see Federal Criteria).

trust
A relationship between two elements, a set of operations and a security policy in which element X trusts element Y if and only if X has confidence that Y behaves in a well defined way (with respect to the operations) that does not violate the given security policy (see ISO/IEC 10081-1).

trusted computing base (TCB)
The totality of protection mechanisms within an IT system, including hardware, firmware, software and data, the combination of which is responsible for enforcing the security policy.

trusted functionality
That which is perceived to be correct with respect to some criteria, for example, as established by a security policy (see ISO/IEC 7498-2).

trusted path
Mechanism by which a person using a terminal can communicate directly with the TCB (see Federal Criteria).

Note: Trusted path can only be activated by the person or the TCB and cannot be imitated by untrusted software.

trusted third party
A security authority or its agent, trusted by other entities with respect to security-related operations (see ISO/IEC 10081-1).

verification AI
Information used by a verifier to verify an identity claimed through exchange AI (see ISO/IEC 10081-2).

verifier
An entity which is or represents the entity requiring an authenticated identity. A verifier includes the functions necessary for engaging in authentication exchanges (see ISO/IEC 10081-2).

vulnerability
Weakness in an information system or components (for example, system security procedures, hardware design, internal controls) that could be exploited to produce an information-related misfortune (see Federal Criteria).

Index