SYS$ACM System Service Functional Specification

Extensible Authentication and Credential Management

Design Team: Rick Barry
Doug Hoeger
Larry Kilgallen
Andy Moskal
Angel Rosario

Revision: V1.0

Date: 21 May 1997

Internal Use Only

The on-line copy of this document can be found at:
STAR::DOCD$:[NT.PROJECT_DOCUMENTS]FS-SYSACM.PS.
Comments can be directed to Rick Barry via|:
e-mail at **barry@star.zko.dec.com**, or
inter-office mail at **ZKO3-4/Y02.**

Revision History

X0.1	11-Mar-1997	Original document.
X0.2	30-Mar-1997	Incorporate comments from internal team review.
X0.3	13-May-1997	Address key issues raised during external review.
V1.0	21-May-1997	Incorporate additional comments from internal review of X0.3.

Associated Documents

- ***Single Sign-On Phase II (SSO-2) External Authentication and Credential Management Service.*** Project Plan.
  Document location: STAR::DOCD$:[NT.PROJECT_DOCUMENTS]PP-SSO-2.PS

- ***Tactical Security Extendible Impersonation Services.*** Design specification.
  Document location: STAR::DOCD$:[NT.PROJECT_DOCUMENTS]DS-NT_TACTICAL_SERVICES.PS

- ***OpenVMS Security Support Provider Interface (SSPI).*** Design specification.
  Document location: STAR::DOCD$:[NT.PROJECT_DOCUMENTS]DS-SSPI.PS

- ***PATHWORKS V6.0B ACME Interface Module for SSO-2.*** Design specification.
  Document location: Not yet available

- ***Windows NT Local Security Authority - Authentication***. November 1, 1994. Microsoft internal document.

- ***Common Internet File System Protocol (CIFS/1.0).*** June 13, 1996. Draft IETF RFC.
  Document location:
  STAR::DOCD$:[NT.PROJECT_DOCUMENTS]DRAFT-HEIZER-CIFS-V1-SPEC-00.TXT

- ***Department of Defense Password Management Guideline***. 12 April, 1985, Department of Defense Computer Security Center, publication CSC-STD-002-85.

- ***OpenVMS Guide to System Security.*** November, 1996. Digital Equipment Corporation.

- ***SecurID Authentication Tokens***
  URL: http://www.securid.com/ID233.22009/Security/tokens.html

- ***CRYPTOCard Challenge/Response Tokens***
  URL: http://www.cryptocard.com/rb1.html

- ***S/Key One-Time Password System***
  URL: ***http://www.bellcore.com/SECURITY/skey.html***

Contents

1. INTRODUCTION 1

1.1 Summary of Requirements 2

2. FUNCTIONAL OVERVIEW 4

2.1 Definition of "Logon" 4

2.2 Domain of Interpretation 4

2.3 Principal Name 4

2.4 Authentication Policy Agents 5

2.5 Authentication and Credential Management Authority 5
2.5.1 SYS$ACM System Service 6
2.5.2 ACM Function Manager 7
2.5.3 ACM Extension (ACME) Agents 7

3. ROUTINE AND DATA SPECIFICATION 9

3.1 $ACM System Service 9
3.1.1 Format 9
3.1.2 C Prototype 9
3.1.3 Arguments 9
efn 9
func 9
context 10
itmlst 13
iosb 15
astadr 15
astprm 16
3.1.4 Function Codes 16
ACME$_FC_AUTHENTICATE_PRINCIPAL 16
ACME$_FC_CHANGE_PASSWORD 19
ACME$_FC_EVENT 20
ACME$_FC_FREE_CONTEXT 20
ACME$_FC_QUERY_DATA 20
ACME$_FC_RELEASE_CREDENTIALS 21
ACME$_FC_SHUTDOWN 22
ACME$_FC_STARTUP 22
3.1.5 Function Modifiers 22
ACME$M_NATIVE_AUTHENTICATION 22
ACME$M_NOAUDIT 22
ACME$M_NOAUTHORIZATION 23
ACME$M_FOREIGN_POLICY_HINTS 23

ACME$M_OVERRIDE_MAPPING 23
ACME$M_TIMEOUT 23
ACME$M_UCS2 23
3.1.6 Item Codes 23
ACME$_ACCESS_PORT 23
ACME$_AUTH_MECHANISM 23
ACME$_CHALLENGE_DATA 24
ACME$_DIALOGUE_SUPPORT 24
ACME$_EVENT_DATA 24
ACME$_LOGON_INFORMATION 24
ACME$_LOGON_TYPE 25
ACME$_MAPPED_VMS_USERNAME 25
ACME$_MAPPING_DOI 25
ACME$_NEW_PASSWORD_1 25
ACME$_NEW_PASSWORD_2 25
ACME$_NEW_PASSWORD_SYSTEM 25
ACME$_PASSWORD_1 26
ACME$_PASSWORD_2 26
ACME$_PASSWORD_SYSTEM 26
ACME$_PERSONA_HANDLE_IN 26
ACME$_PERSONA_HANDLE_OUT 26
ACME$_PRINCIPAL_NAME_IN 26
ACME$_PRINCIPAL_NAME_OUT 26
ACME$_QUERY_DATA 26
ACME$_QUERY_DOI 26
ACME$_QUERY_KEY 26
ACME$_QUERY_TYPE 26
ACME$_REMOTE_HOST_ADDRESS 27
ACME$_REMOTE_HOST_ADDRESS_TYPE 27
ACME$_REMOTE_HOST_NAME 27
ACME$_REMOTE_USER_NAME 27
ACME$_RESPONSE_DATA 27
ACME$_TARGET_DOI 27
ACME$_TIMEOUT_INTERVAL 27
ACME$_CHAIN 27
ACME$_IGNORE 28
3.1.7 Description 28
Modes of Operation 28
Non-dialogue (Non-iterative) Mode 28
Dialogue (Iterative) Mode 29
Privileges and Restrictions 30
3.1.8 Condition Values Returned in R0 30
3.1.9 Condition Values Returned in the I/O Status Block 31

3.2 ACME Agent Requirements 32

4. OPENVMS SSPI OPERATION 33

5. IMPACT ON EXISTING BASE COMPONENTS 34

5.1 SYS$SINGLE_SIGNON Bit-Mask Controls 34

5.2 LOGINOUT 34

5.3 SET PASSWORD 34

5.4 Authorize and $SETUAI/$GETUAI 35

6. ISSUES 36

6.1 Open Issues 36

6.2 Resolved Issues 36

7. INTERACTIVE AUTHENTICATION PROTOTYPES 39

7.1 SecurID Token 39

7.2 CRYPTOCard RB-1 Token 39

7.3 S/Key One-Time Password System 40

8. WINDOWS NT-SPECIFIC INFORMATION 41

8.1 MSV1_0 Credentials 41

8.2 MSV1_0 Logon Input/Output Parameters 41

8.3 NT_LM ACME Constants and Item Codes 42

9. GLOSSARY 43

1. Introduction

This functional specification defines a service interface for an OpenVMS extensible user authentication mechanism. A new *Authentication and Credential Management (ACM) Authority,* whose role will be played by the security server, provides a back-end interface to replaceable plug-in modules and a standard front-end application programming interface.

The ACM authority serve two major functions:

1. The $ACM system service presents a standard programming interface for user authentication. For interactive authentication, the service controls the prompting sequence needed to accomplish authentication. It does this by requiring the caller to make iterative calls to the service until all necessary data is received. For non-interactive authentication, all input will typically be predetermined and supplied in a single service call. In either case, user account modal restrictions are enforced, intrusion detection is performed, and the proper audits are issued. This ensures that any code that calls this service (OpenVMS login process, DECwindows pause screen, FTP server daemon, etc.) will enforce a consistent authentication policy. Plug-in modules provide support for multiple authentication policies: OpenVMS, Windows NT, and so on.

   The service also provides a standard interface for user password management.

2. The ACM authority merges extended persona support and authentication to provide integrated credential management with user authentication. Normal persona support allows code to obtain native (OpenVMS) process credentials, such as *username*, *UIC*, and *rights identifiers*. In addition to this, extended persona support enables a process to obtain non-native credentials. This would, as an example, allow a process to possess both Windows NT credentials and OpenVMS credentials (see Figure 1).

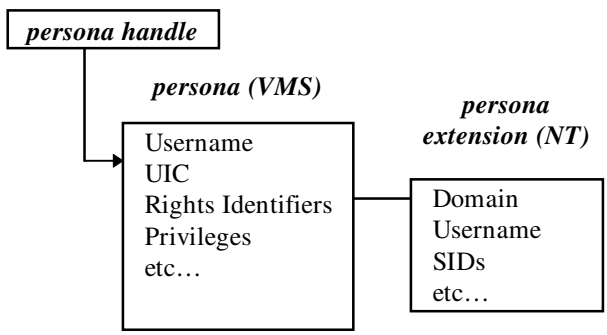


Figure 1 Persona and persona extension

The interface is designed for flexibility to support commonly available authentication mechanisms. (See section 7, for a list of interactive authentication prototypes that were considered during the design.) It is also intended to be extendible to allow future capabilities to be added in a forward compatible manner with minimal effort.

The primary consumers of this service are the OpenVMS login process (LOGINOUT), the SET PASSWORD command, and NT Infrastructure's Security Support Provider Interface (SSPI). See Figure 2. Other code layered on OpenVMS is also expected to take advantage of consistent authentication features provided by this service.

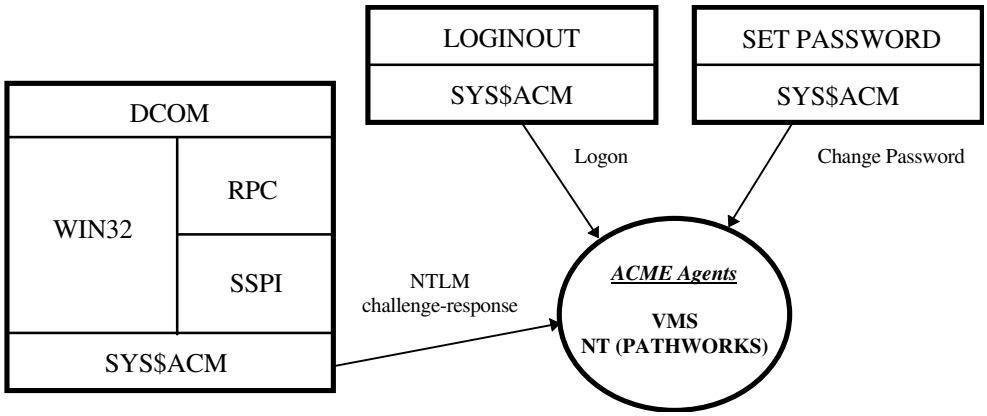


Figure 2 $ACM consumers

1.1 Summary of Requirements

This section summarizes the requirements addressed within this specification. For a detailed requirements list, please refer to the Single Sign-On Phase II (SSO-2) project plan.

NT Infrastructure requirements:

1. Establish NT 4.0-style credentials for a process or user thread upon successful authentication by an NT domain controller.
2. Provide NT 4.0-style authentication for interactive user logon using domain, username, and password. Return user information specific to an interactive logon.
3. Provide NT 4.0-style authentication for network logon using domain, username, workstation, and challenge response. Return user information specific to a network logon.
4. Provide the means for a user to change his or her NT password and have it updated by the primary domain controller.
5. Provide forward and reverse translations for the following NT security attributes:
    - Account SID and Account Name
    - Privilege LUID and Privilege Name
6. Provide a channel to the PATHWORKS server for generating an access audit event.
7. Allow for UNICODE representation of strings.

OpenVMS requirements:

1. Provide native OpenVMS authentication based on username and password(s).
2. Provide native logon modal restrictions, intrusion detection, and login/logout/logfail auditing.
3. Provide native password management, including generated passwords, password dictionary, password history, and password filter.

General single sign-on requirements:

1. Allow the native username + password based authentication policy to be replaced or augmented by alternative mechanisms (see page 39).
2. Support multiple authentication policies.

3. Return account information, such as process quotas, upon successful authentication..

2. Functional Overview

This is an overview of the ACM model and the features it provides in order to satisfy the requirements stated in section 1.1.

2.1 Definition of "Logon"

The ACM model defines a "logon" to be a successful authentication procedure that results in a "principal" (user or process) gaining some type of access to the OpenVMS operating system. There are two classes of logon: **interactive** and **non-interactive**. Each may have specific policy requirements. For example, an interactive logon may be authenticated differently than a non-interactive logon depending on the policy.

Logon types:

interactive:	*local*, *dialup*, *remote*
non-interactive:	*detached, network*

Note: This service is not intended to replace distributed authentication mechanisms, such as those available through the GSS-API or SSPI interfaces. Its two main goals are: 1) to provide a standard programming interface to common authentication for the purpose of OpenVMS system logons, and 2) for managing credentials needed to enforce security policy. In the case of Windows NT, this service works in conjunction with the SSPI to provide NTLM distributed security.

2.2 Domain of Interpretation

A ***domain of interpretation*** (DOI) defines the authentication policy that is enforced by a particular authentication agent and the type of credentials it provides (see section 2.3). Each domain of interpretation has associated with it a name that is used to identify it when interacting with the $ACM service.

Two DOIs are planned for Raven:

OpenVMS domain of interpretation:	"**VMS**"
Windows NT domain of interpretation:	"**NT_LM**"

2.3 Principal Name

A ***principal*** is any entity that is subject to authentication. A principal is identified by its unique name within the domain of interpretation in which it belongs. The syntax and character set of a principal name is defined by its domain of interpretation, but the ACM model requires the principal name to be entirely represented using a single string name. The DOI is responsible for defining the syntax of this string in any manner suitable to its particular needs.

The following describes the syntax of a principal name for the planned DOIs:

DOI	Principal Name Syntax
VMS	*username*
NT_LM	*username@domain*

In the case of NT_LM DOI, the principal name reflects a user account within a LAN Manager domain.

2.4 Authentication Policy Agents

The policy for a given domain of interpretation is implemented by an authentication agent. The DOI authentication agent for OpenVMS enforces native authentication policy; other DOI authentication agents enforce external authentication policies. Section 2.5 describes the ACM authority and how it supports different DOIs.

The ACM model allows external authentication policies, such as the Windows NT authentication policy, to coexist with the native OpenVMS authentication policy. By design, the external authentication policy does not completely replace the native policy. This is partly because native credentials are still required to perform access checks on native objects. To this end, an external user profile identified by principal name is always associated with a native user profile identified by an OpenVMS username. This is also done to permit local system management to maintain control over the local user accounts that are used to map external principals to OpenVMS usernames.

In general, the ACM model allows an external authentication policy to…

1. …*replace* the native authentication password-based policy, including password expiration, minimum/maximum password length, password history, and password filters.
2. …*augment* modal restrictions (hour-of-day, day-of-week). Native modal checks are still enforceable.
3. …*augment* intrusion detection. Native intrusion detection is still enforceable.
4. …*augment* auditing. Native login/logfail auditing may still occur.

External authentication policy *does not override* OpenVMS account controls (such as **DISUSER**, **CAPTIVE**, and **RESTRICTED** flags), quotas, or privileges.

2.5 Authentication and Credential Management Authority

OpenVMS native and external authentication policy is totally controlled by the ACM authority and its extension agents. The ACM authority functions will be provided by the OpenVMS Security Server with enhancements to implement the features described in this document.

The design partitions the ACM authority into three logical components:

1. $ACM system service front-end
2. ACM function manager (routing)
3. One or more back-end, local ACM extension agents (ACME agents)

Figure 3 Figure 3 contains a diagram of the ACM authority components. The $ACM service is responsible for presenting a consistent interface to software while the ACM function manager is responsible for dispatching function requests/responses to and from the ACME agents. It is the ACME agent that contains policy information specific to its domain of interpretation (DOI) and is responsible for enforcing policy and providing appropriate credentials.

For Raven, only two ACME agents are planned: OpenVMS and NT. Other ACME agents can be implemented and plugged into the ACM function manager in the future as each module is simply a shareable image written to an ACME-compliant interface.

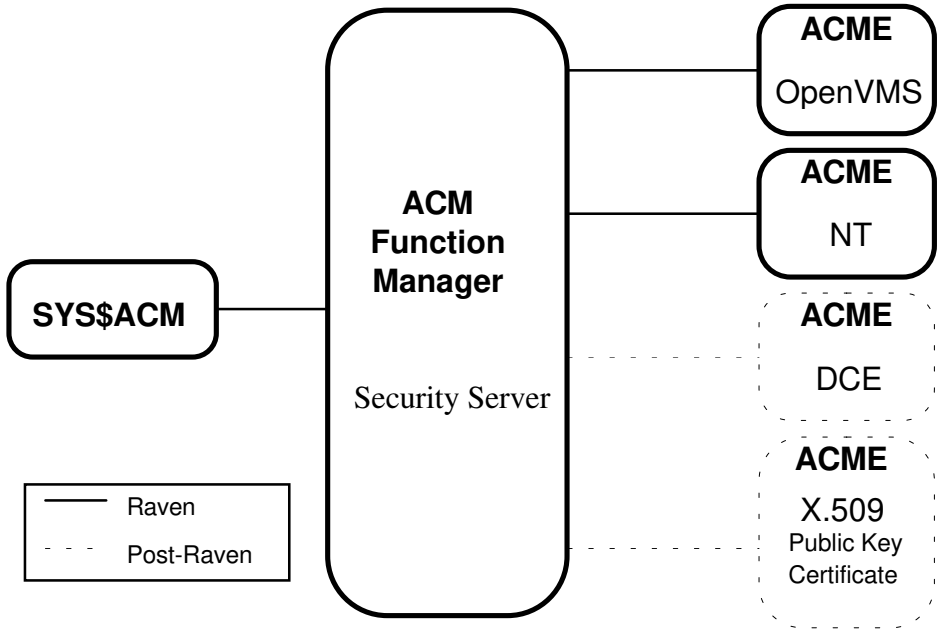


Figure 3: ACM Authority

2.5.1 SYS$ACM System Service

As already mentioned, the $ACM service is used by code to perform user authentication. For the greatest flexibility and future extendibility, this service is function-based and all input/output is specified through an item list. In addition, the item codes are partitioned into common and ACME-specific spaces to allow ACME agents to describe their own data without the worry of item code conflicts. In most cases, the caller of $ACM does not need to understand these ACME-specific item codes since they are transparently received from the service and then sent back to the service on the next iteration (see section 3.1.7). Only those callers that have specific knowledge about the particular ACME will explicitly use them.

2.5.1.1 Integrated Persona Management

The $ACM authentication function is integrated with the OpenVMS persona services for the convenient management of credentials. When a caller needs to obtain credentials, the $ACM authentication function can operate in one of three ways:

1. **New persona**. This is the default mode of operation and returns a handle to a new persona containing the principal's OpenVMS credentials and any external credentials.

2. **Copy persona**. This operation copies an existing persona. It then merges the acquired external credentials with that persona and returns its handle. The acquired external credentials are added to or replace the existing credentials for that domain of interpretation. The OpenVMS credentials are not changed. If no persona handle is provided the active persona is used.

3. **Merge persona**. This operation merges the acquired external credentials with an existing persona and returns a handle to that persona. The credentials add to or replace the existing credentials for

that domain of interpretation. The OpenVMS credentials are not changed. If no persona handle is provided the active persona is used.

The default action is to return an error if different domains of interpretation map to different OpenVMS accounts, but applications can override this behavior.

$ACM does not implicitly assume a persona. It is up to the caller to assume any persona created or modified by $ACM.

2.5.2 ACM Function Manager

The ACM function manager is responsible for routing $ACM requests and responses between the caller and the ACME agent. Requests are posted by the $ACM service to a pending queue which are then serviced by the ACM function manager and dispatched to the appropriate ACME agent for processing. Once the ACME agent has finished processing the request, it posts it to the finished queue where the ACM function manager completes the request by scheduling a kernel AST to the caller that enables the $ACM service to complete. See Figure 4.

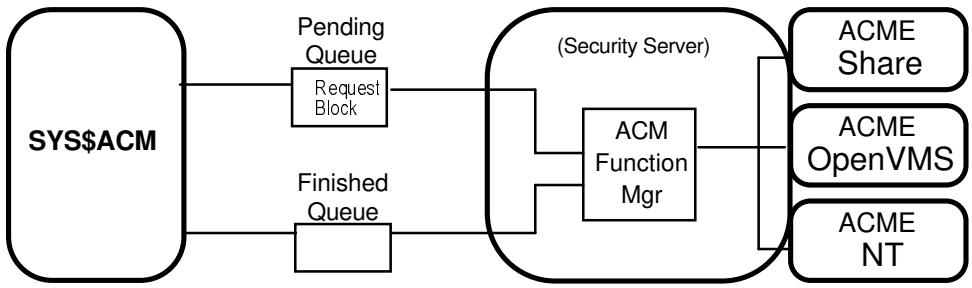


Figure 4: Communication among ACM authority components

The ACM function manager is mostly unaware of the contents of the request packets. Its main functions are routing requests and basic error logging.

2.5.2.1 Error Logging

As an aid in diagnosing configuration problems, the ACM function manager will provide rudimentary error logging. (ACME agents may optionally provide their own error logging in addition to what the ACM function provides.) The logging mechanism will be based on OPCOM messages. Error logging can be dynamically turned on or off. This will be accomplished using the existing SYS$SINGLE_SIGNON logical name bit-mask controls.

The error log will contain information about the request itself, the final status, and the source of the request.

2.5.3 ACM Extension (ACME) Agents

Each domain of interpretation must have an associated ACME agent that is responsible for actually carrying out the various operations needed to satisfy a particular function request and for enforcing the authentication policy defined for its domain of interpretation. The ACME agent code is an ACME interface-compliant shareable image that is activated and run within the context of the security server.

The security server will activate the ACME agent images in the order specified by the system manager. The order of activation also specifies the order in which principal name mapping attempts occur.

For example, if the ACME agent ordering is "VMS", “NT_LM", then the OpenVMS ACME agent will be given the first chance of mapping a principal name. If that mapping fails, then the NT_LM ACME agent is called to attempt a mapping.

Note: The “VMS”, “NT_LM” ordering is not static and can be configured differently depending on particular needs.

3. Routine and Data Specification

The section describes the calling interface, functions, function modifiers, item codes, and associated data structures related to the $ACM service.

3.1 $ACM System Service

The $ACM service provides a common interface to all functions supported by the Authentication and Credentials Management (ACM) authority.

The caller must specify the function code and any applicable function modifiers and item codes for the requested operation.

The $ACM service completes asynchronously; for synchronous completion, use the $ACMW form of the service.

3.1.1 Format

SYS$ACM[W] [efn], func, [context], itmlst, iosb, [astadr], [astprm]

3.1.2 C Prototype

```
int sys$acm[w]  ( unsigned int efn, unsigned int func, struct _acmecb *context,
                  struct _iosb *iosb, void (*astadr)(__unknown_params), int astprm);
```

3.1.3 Arguments

efn

OpenVMS usage: ef_number

type: longword (unsigned)

access: read only

mechanism: by value

Number of the event flag to be set when the $ACM request completes. The **efn** argument is a longword containing this number; however, $ACM uses only the low-order byte.

When the request is queued, $ACM clears the specified event flag. Then, when the request completes, the specified event flag is set.

func

OpenVMS usage: function_code

type: longword (unsigned)

access: read only

mechanism: by value

Function code and modifiers specifying the operation $ACM is to perform. The **func** argument is a longword containing the function code.

Function codes have symbolic names of the following format:

ACME$_FC_code

Function modifiers have symbolic names of the following format:

ACME$M_modifier

The $ACMEDEF macro defines the names of each function code and modifier.

Only one function code can be specified across a sequence of related calls to $ACM. Most function codes require or allow for additional information to be passed in the call. This information is passed using the **itmlst** argument, which specifies a list of one or more item descriptors. Each item descriptor in turn specifies an item code, which either controls the action to be performed or requests specific information to be returned.

Refer to section 3.1.7 for further information regarding the iterative nature of the dialogue mode of operation.

context

OpenVMS usage:	context
type:	longword (unsigned)
access:	modify
mechanism:	by 32- or 64-bit reference (Alpha)
mechanism:	by 32-bit reference (VAX)

Address of a longword to receive the 32-bit address of an ACM communications buffer. The **context** argument is used by $ACM during dialogue mode to request additional information needed to complete a request.

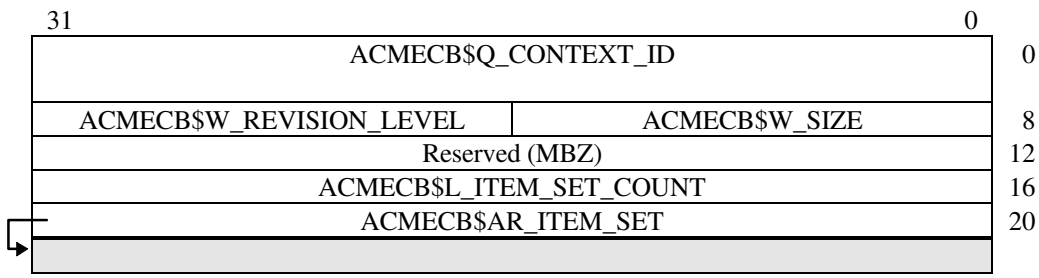
If the **context** argument is unspecified or 0 and additional information is required, $ACM returns a failure status.

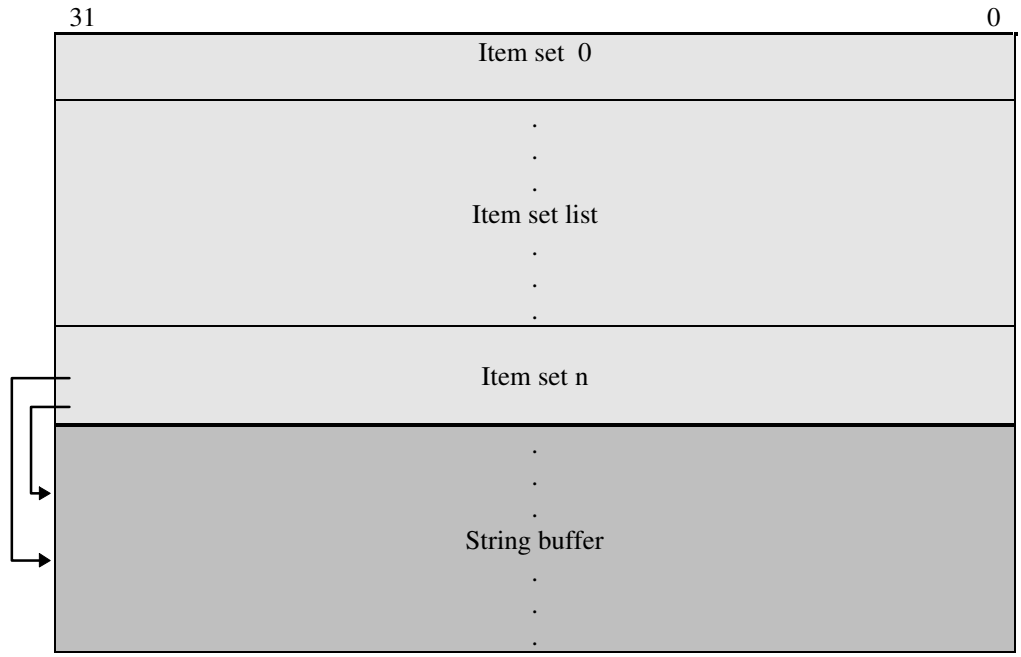
The longword specified must contain a -1 on the first call in a sequence of related calls . If additional information is required to complete the request, $ACM returns a pointer to an ACM communications buffer which describes what information is needed. The buffer address must be passed back on the subsequent call to $ACM to continue with the particular operation. The requested information is passed back in the subsequent call using the **itmlst** argument.

To terminate a request and release the associated buffer, use the ACME$_FC_FREE_CONTEXT function specifying the address of the ACM communications buffer using the **context** argument.

The ACM communications buffer is user readable, but not user writeable. It consists of a structure header segment, an item set list segment and a string buffer segment. The item set list segment is an array of item set entries, each of which describes an item of information that is needed to complete the request. The item set entries are processed sequentially in the order in which they reside in the list.

The following diagram depicts the overall format of an ACM communications buffer.

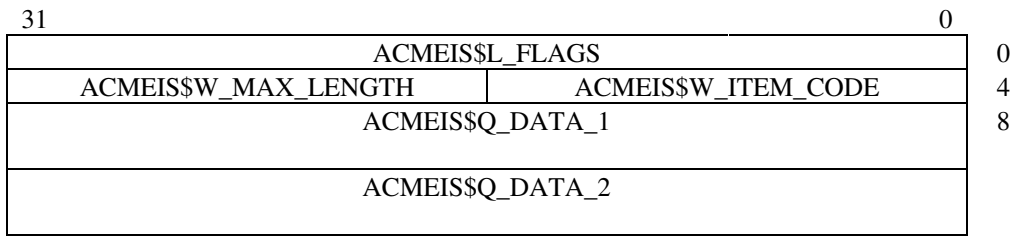




The following table defines the ACM communications buffer header fields.

Descriptor Field	Definition
ACMECB$Q_CONTEXT_ID	A quadword containing a value used internally by $ACM to uniquely bind the ACM communications buffer to the associated request across iterative calls.
ACMECB$W_SIZE	A word containing the size of the ACM communications structure.
ACMECB$W_REVISION_LEVEL	A word containing a value which identifies the revision level of the ACM communications structure.
ACMECB$L_ITEM_SET_COUNT	A longword containing the number of item sets in the item set list.
ACMECB$AR_ITEM_SET	A longword containing the 32-bit address of the item set list. The item set list is an array of item set entries which describe the information needed by $ACM to complete the particular request.

The following diagram depicts the format of an item set entry.



The following table defines the item set entry fields.

Descriptor Field	Definition

Descriptor Field	Definition
ACMEIS$L_FLAGS	A longword containing a mask of flags which indicate the interpretation of the ACMEIS$Q_DATA_1 and ACMEIS$Q_DATA_2 fields and method for processing. Each flag has a symbolic name, the $ACMEDEF macro defines these names. The following list defines the flags. ACMEIS$V_INPUT When clear • ACMEIS$Q_DATA_1 and ACMEIS$Q_DATA_2 represent text to be displayed to the caller. A length of 0 in either descriptor indicates the particular data field is to be ignored. When set • ACMEIS$W_ITEM_CODE defines the item code to use when providing the requested information. • ACMEIS$Q_DATA_1 represents prompt text. A length of 0 indicates no specific prompt text. • Interpretation of ACMEIS$Q_DATA_2 is determined by the state of the ACMEIS$V_NOECHO flag. ACMEIS$V_NOECHO When clear • ACMEIS$Q_DATA_2 represents the default response. A length of 0 indicates no default response. When set • The associated input data is not to be echoed. This is typically used to visually protect sensitive data, such as password information. • ACMEIS$Q_DATA_2 represents secondary prompt text for input verification processing. A length of 0 indicates no verification processing. This flag is ignored when ACMEIS$V_INPUT is clear.
ACMEIS$W_ITEM_CODE	A word containing the item code which identifies the nature of information associated with the item set. This field defines the item code to use when specifying the requested information in the **itmlst** argument in the subsequent call to $ACM. A sequence of item sets containing the same item code reflects a linked list of related information. An item set with the ACMEIS$V_INPUT flag set indicates the end of the related list.

Descriptor Field	Definition
ACMEIS$W_MAX_LENGTH	A word containing the maximum length of any input data being requested by this item set. For example, the maximum length of a principal name string. A value of 0 implies no specific limit from the service perspective. This field is ignored when ACMEIS$V_INPUT is clear.
ACMEIS$Q_DATA_1	A quadword containing a 32-bit descriptor which describes information (such as prompt text) applicable to determining an appropriate response.
ACMEIS$Q_DATA_2	A quadword containing a 32-bit descriptor which describes information (such as a default answer) applicable to determining an appropriate response.

itmlst

OpenVMS usage: 32-bit item_list_3 or 64-bit item_list_64b

type: longword (unsigned) for 32-bit; quadword (unsigned) for 64-bit

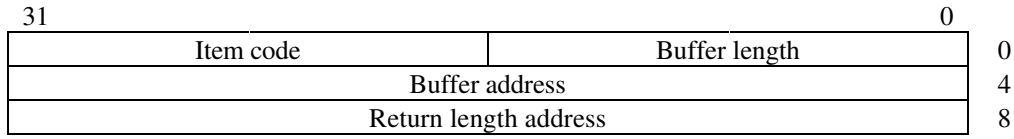
access: read only

mechanism: by 32- or 64-bit reference (Alpha)

mechanism: by 32-bit reference (VAX)

Item list specifying information to be used in performing the function and requesting information to be returned. The **itmlst** argument is the 32-bit address (on VAX systems) and the 32- or 64-bit address (on Alpha systems) of a list of item descriptors, each of which describes an item of information. An item list in 32-bit format is terminated by a longword of 0; an item list in 64-bit format is terminated by a quadword of 0. All items in an item list must be of the same format - either 32-bit or 64-bit.

The following diagram depicts the 32-bit format of a single item descriptor.

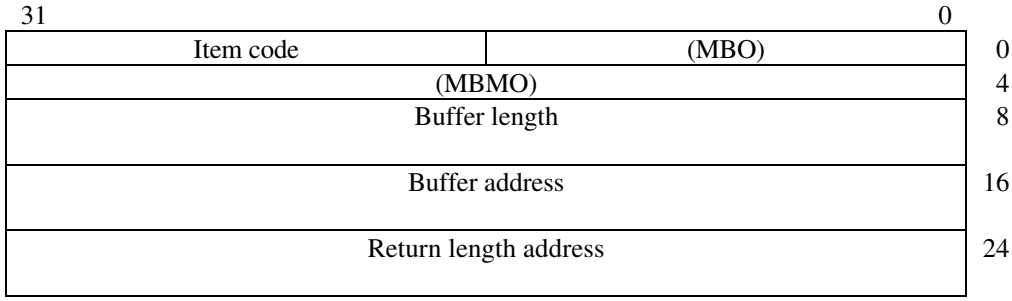


The following table defines the item descriptor fields for 32-bit item list entries.

Descriptor Field	Definition
Buffer length	A word specifying the length of the buffer (in bytes); the buffer either supplies information to $ACM or receives information from $ACM. The required length of the buffer varies, depending on the item code specified, and is given in the description of each item code.
Item code	A word containing an item code, which identifies the nature of the information supplied to $ACM or which is received from $ACM. Each item code has a symbolic name; the $ACMEDEF macro defines these symbolic names.

Descriptor Field	Definition
Buffer address	A longword containing the 32-bit address of the buffer that specifies or receives the information.
Return length address	A longword containing the 32-bit address of a word to receive the length (in bytes) of information returned by $ACM. If specified as 0, no length is returned.

The following diagram depicts the 64-bit format of a single item descriptor.



The following table defines the item descriptor fields for 64-bit item list entries.

Descriptor Field	Definition
MBO	A word which must contain a 1. The MBO and MBMO fields are used to distinguish 32-bit and 64-bit item list entries.
Item code	A word containing an item code, which identifies the nature of the information supplied to $ACM or which is received from $ACM. Each item code has a symbolic name; the $ACMEDEF macro defines these symbolic names.
MBMO	A longword which must contain a -1. The MBMO and MBO fields are used to distinguish 32-bit and 64-bit item list entries.
Buffer length	A quadword specifying the length of the buffer (in bytes); the buffer either supplies information to $ACM or receives information from $ACM. The required length of the buffer varies, depending on the item code specified, and is given in the description of each item code.
Buffer address	A quadword containing the 64-bit address of the buffer that specifies or receives the information.
Return length address	A quadword containing the 64-bit address of a quadword to receive the length (in bytes) of information returned by $ACM. If specified as 0, no length is returned.

Item codes have symbolic names of the following format:

ACME$_code

The $ACMEDEF macro defines the names of each function code and modifier.

The are two types of item code:

- **Input item code**. Input item codes specify a buffer which contains information provided to $ACM.. The buffer length and buffer address fields in the item descriptor must be non-zero; the return length field must be zero.
- **Output item code**. Output item codes specify a buffer in which $ACM is to return information. The buffer length and buffer address fields of the item descriptor must be non-zero; the return length field can be zero or non-zero.

Refer to section 3.1.6 for a description of the various item codes.

iosb

OpenVMS usage: io_status_block

type: quadword (unsigned)

access: write only

mechanism: by 32- or 64-bit reference (Alpha)

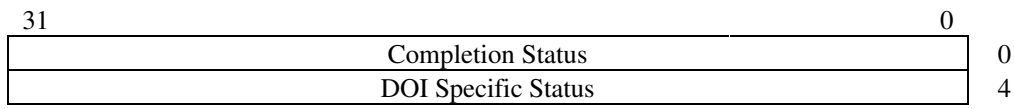
mechanism: by 32-bit reference (VAX)

I/O status block that is to receive the final completion status. The **iosb** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) of the quadword I/O status block.

Upon request initiation, $ACM sets the value of the quadword I/O status block to 0 . When the requested operation completes, $ACM writes a condition value in the first longword of the I/O status block. The condition value indicates whether the operation completed successfully or failed.

When failing a request, an ACME agent can return a longword value which indicates additional information about the failure. $ACM writes this value in the second longword of the I/O status block. Interpretation of the value may be specific to the target (mapping) domain of interpretation and typically reflects a status code specific to that context.

The following diagram depicts the structure of an I/O status block.



The following table defines the I/O status block fields.

Descriptor Field	Definition
Completion Status	A longword containing the completion status for the requested operation.
DOI Specific Status	A longword containing additional status information. The interpretation of this value is specific to the context of the target (mapping) domain of interpretation.

astadr

OpenVMS usage: ast_procedure

type: procedure value

access: call without stack unwinding

mechanism: by 32- or 64-bit reference (Alpha)

mechanism: by 32-bit reference (VAX)

AST service routine to be executed when $ACM completes. The **astadr** argument is the 32-bit address (on VAX systems) or the 32- or 64-bit address (on Alpha systems) of this routine.

The AST routine executes at the same access mode as the caller of the $ACM service.

astprm

OpenVMS usage: user_arg

type: longword (unsigned)

access: read only

mechanism: by value

AST parameter to be passed to the AST service routine specified by the **astprm** argument. The **astprm** argument is the longword parameter.

3.1.4 Function Codes

This section describes the various function codes supported by the $ACM service. It lists the applicable function modifiers and item codes for each function.

Refer to sections 3.1.5 and 3.1.6 for further information regarding the various function modifiers and item codes. Refer section 3.1.7 for information relating to the modes of operation and privilege requirements.

General Functions

ACME$_FC_AUTHENTICATE_PRINCIPAL

The ACME$_FC_AUTHENTICATE_PRINCIPAL function authenticates a principal name within a particular domain of interpretation and optionally returns credentials following successful authorization.

The principal name is specified using the ACME$_PRINCIPAL_NAME_IN item code. If not provided on the first call to $ACM, the domains of interpretation are queried to determine if principal name prompting or auto-authorization processing should be initiated. The domains of interpretation are queried in the order configured by the system manager.

Authentication requests can be directed towards a specific domain of interpretation using the ACME$_TARGET_DOI item code. If not explicitly provided, authentication proceeds under the auspices of the first domain of interpretation to recognize the principal name. In this situation, domains of interpretation are queried in the order configured by the system manager. If no domain of interpretation recognizes the principal name, authentication ultimately fails.

When the authenticating domain of interpretation is not “VMS”, the domain of interpretation provides an OpenVMS username to which the principal name maps. The OpenVMS username must reflect a valid account record in the OpenVMS authorization file (SYSUAF.DAT) which has the UAF$V_EXTAUTH flag set. The OpenVMS username is used as the basis for supplemental authorization and determining process parameters within the “VMS” domain of interpretation.

The ACME$M_ACQUIRE_CREDENTIALS function modifier is used to obtain credentials following successful authorization.

Only the “VMS” domain of interpretation provides credentials on the VAX platform.

The ACME$_PERSONA_HANDLE_OUT item code is used to obtain a handle to the persona upon which credential related operations were executed. The ACME$_PERSONA_HANDLE_IN item code is used to specify an existing persona for credential related operations.

The default action is to create a new persona. The following function modifiers alter that behavior.

ACME$M_MERGE_PERSONA	Attach any external credentials to the specified persona
ACME$M_COPY_PERSONA	Attach any external credentials to a copy of the specified persona

The following table depicts the effect these function modifiers have on persona processing.

Operation Modifier	Input Handle	Persona Used	Output Handle
New Persona	N/A	N/A	New Persona
Merge Persona	Null	Active Persona	Active Persona
Merge Persona	Specified Persona	Specified Persona	Specified Persona
Copy Persona	Null	Active Persona	New Persona
Copy Persona	Specified Persona	Specified Persona	New Persona

When a new persona is created, the credentials that resulted in the creation of the persona are designated as the “primary credentials”.

To operate in dialogue mode, the **context** argument must be specified.

The following function modifiers can be specified.

ACME$M_NATIVE_AUTHENTICATION

ACME$M_NOAUDIT

ACME$M_NOAUTHORIZATION

ACME$M_OVERRIDE_MAPPING

ACME$M_FOREIGN_POLICY_HINT

ACME$M_TIMEOUT

ACME$M_UCS2

The following item codes can be specified in the initial call for an authenticate principal request.

ACME$_ACCESS_PORT

ACME$_AUTH_MECHANISM

ACME$_CHALLENGE

ACME$_DIALOGUE_SUPPORT

ACME$_LOGON_INFORMATION

ACME$_LOGON_TYPE

ACME$_MAPPED_VMS_USERNAME

ACME$_MAPPING_DOI

ACME$_PASSWORD_1

ACME$_PASSWORD_2

ACME$_PASSWORD_SYSTEM

ACME$_PERSONA_HANDLE_IN

ACME$_PERSONA_HANDLE_OUT

ACME$_PRINCIPAL_NAME_IN

ACME$_PRINCIPAL_NAME _OUT

ACME$_QUERY_DOI

ACME$_QUERY_DATA

ACME$_QUERY_KEY

ACME$_QUERY_TYPE

ACME$_REMOTE_HOST_ADDRESS

ACME$_REMOTE_HOST_ADDRESS_TYPE

ACME$_REMOTE_HOST_NAME

ACME$_REMOTE_USER_NAME

ACME$_RESPONSE

ACME$_TARGET_DOI

ACME$_TIMEOUT_INTERVAL

If the ACME$_CHALLENGE item code is specified, the ACME$_AUTH_MECHANISM and ACME$_RESPONSE item codes must also be specified.

The following status codes can be returned in the I/O status block.

ACME$_NORMAL

ACME$_ACCOUNTLOCK

ACME$_AUTHFAILURE

ACME$_BUFFEROVF

ACME$_DOIUNAVAILABLE

ACME$_INTRUDER

ACME$_INVALIDCTX

ACME$_INVALIDPWD

ACME$_INVPERSONA

ACME$_INVREQUEST

ACME$_MAPCONFLICT

ACME$_NOAUTH

ACME$_NOEXTAUTH

ACME$_NOPRIV

ACME$_NOSUCHDOI

ACME$_NOSUCHUSER

ACME$_OPINCOMPL

ACME$_TIMEOUT

ACME$_UNSUPPORTED

ACME$_FC_CHANGE_PASSWORD

The ACME$_FC_CHANGE_PASSWORD function performs a password change operation. The primary and secondary passwords may be changed independently.

If the SYS$SINGLE_SIGNON logical name bit-mask control for password synchronization is ON and the target domain of interpretation is not “VMS”, then the password will be copied to the SYSUAF.DAT account record mapped for the user, unless the UAF$V_PASSWORDSYNCH flag is OFF.

The following function modifiers can be specified.

ACME$M_TIMEOUT

ACME$M_UCS2

The following item codes can be specified in the initial call for a change password request.

ACME$_DIALOGUE_SUPPORT

ACME$_MAPPED_VMS_USERNAME

ACME$_MAPPING_DOI

ACME$_NEW_PASSWORD_1

ACME$_NEW_PASSWORD_2

ACME$_NEW_PASSWORD_SYSTEM

ACME$_PASSWORD_1

ACME$_PASSWORD_2

ACME$_PASSWORD_SYSTEM

ACME$_PRINCIPAL_NAME_IN

ACME$_PRINCIPAL_NAME_OUT

ACME$_TARGET_DOI

ACME$_TIMEOUT_INTERVAL

If either ACME$_PRINCIPAL_NAME_IN or ACME$_TARGET_DOI is specified, then both must be specified.

The following status codes can be returned in the I/O status block.:

ACME$_NORMAL

ACME$_ACCOUNTLOCK

ACME$_DOIUNAVAILABLE

ACME$_INVALIDCTX

ACME$_INVALIDPWD

ACME$_INVREQUEST

ACME$_MAPCONFLICT

ACME$_NOEXTAUTH

ACME$_NOPRIV

ACME$_NOSUCHDOI

ACME$_NOSUCHUSER

ACME$_OPINCOMPL

ACME$_TIMEOUT

ACME$_UNSUPPORTED

ACME$_FC_EVENT

The ACME$_FC_EVENT function provides a simple logging feature that can be used to generate certain events related to the policy of a domain of interpretation. To log an event, supply the desired "event" item code followed by the appropriate "data" item code pertaining to the domain of interpretation.

In Raven, the only support provided is to log a Windows NT access-checking event. Refer to section 8.3 for NT-specific event/data item codes

The following function modifiers can be specified.

ACME$M_TIMEOUT

ACME$M_UCS2

The following item codes must be specified.

ACME$_EVENT_DATA

ACME$_TARGET_DOI

The following item codes can be specified.

ACME$_TIMEOUT_INTERVAL

The following status codes can be returned in the I/O status block.

ACME$_NORMAL

ACME$_DOIUNAVAILABLE

ACME$_INVREQUEST

ACME$_NOPRIV

ACME$_NOSUCHDOI

ACME$_TIMEOUT

ACME$_UNSUPPORTED

ACME$_FC_FREE_CONTEXT

The ACME$_FC_FREE_CONTEXT function is used to terminate iterative processing of a request. The address of the ACM communications buffer associated with the request must be specified using the **context** argument.

The following status codes can be returned in the I/O status block.

ACME$_NORMAL

ACME$_INVALIDCTX

ACME$_FC_QUERY_DATA

The ACME$_FC_QUERY_DATA function provides a simple key-based query feature that can be used to obtain certain information related to the policy of a domain of interpretation. To lookup up an item of information, supply the desired “key” item code followed by the appropriate “data” item code.

In Raven, the only support provided is to query the SAM database to obtain SID-to-name and privilege LUID-to-privilege name mapping information and to calculate and return challenge-response information. Refer to section 8.3 for NT-specific key/data item codes

The following function modifiers can be specified.

ACME$M_TIMEOUT

ACME$M_UCS2

The following item codes must be specified.

ACME$_QUERY_DOI

The following item codes can be specified.

ACME$_QUERY_DATA

ACME$_QUERY_KEY

ACME$_QUERY_TYPE

ACME$_TIMEOUT_INTERVAL

The following status codes can be returned in the I/O status block.

ACME$_NORMAL

ACME$_DOIUNAVAILABLE

ACME$_INVREQUEST

ACME$_NOPRIV

ACME$_NOSUCHDOI

ACME$_TIMEOUT

ACME$_UNSUPPORTED

ACME$_FC_RELEASE_CREDENTIALS

The ACME$_FC_RELEASE_CREDENTIALS function removes credentials for a particular domain of interpretation from the specified persona. All non-native credentials are released and the persona is deleted when the domain of interpretation is specified as “VMS”. The “VMS” credentials cannot be removed from either the currently active or process’ natural persona.

The following item codes must be specified.

ACME$_PERSONA_HANDLE_IN

ACME$_TARGET_DOI

The following status values can returned in the I/O status block.

ACME$_NORMAL

ACME$_INVPERSONA

ACME$_NOPRIV

ACME$_NOSUCHDOI

ACME$_NOTARGETCREDS

Management Functions

ACME$_FC_SHUTDOWN

The ACME$_FC_SHUTDOWN function requests the ACM authority to deactivate the specified ACME agent. This disables operations to the domain of interpretation of interpretation associated with the ACME agent.

The OpenVMS ACME agent cannot be deactivated.

The following item codes must be specified.

ACME$_TARGET_DOI

The following status codes can be returned in the I/O status block.

ACME$_NORMAL

ACME$_DOIUNAVAILABLE

ACME$_NOPRIV

ACME$_NOSUCHDOI

ACME$_FC_STARTUP

The ACME$_FC_STARTUP function requests the ACM authority to activate the specified ACME agent. The OpenVMS ACME agent is always started by the ACM authority during system startup.

The following item codes must be specified..

ACME$_TARGET_DOI

The following status codes can be returned in the I/O status block.

ACME$_NORMAL

ACME$_DOIUNAVAILABLE

ACME$_NOPRIV

ACME$_NOSUCHDOI

3.1.5 Function Modifiers

This section describes the various function modifiers applicable to the function codes supported by the $ACM service

ACME$M_NATIVE_AUTHENTICATION

The ACME$M_NATIVE_AUTHENTICATION function modifier forces a mapping to be made to an OpenVMS username regardless of the UAF$V_EXTAUTH flag setting for that account. This modifier is used to support the /LOCAL_PASSWORD qualifier of LOGINOUT which allows native authentication to be performed to an OpenVMS user account which would normally be authenticated by a non-native domain of interpretation (UAF$V_EXTAUTH flag is set).

ACME$M_NOAUDIT

The ACME$M_NOAUDIT function modifier indicates auditing actions should not be performed. Unless the ACME$M_FOREIGN_POLICY_HINTS function modifier is also specified, this modifier applies only to the native domain of interpretation.

ACME$M_NOAUTHORIZATION

The ACME$M_NOAUTHORIZATION function modifier indicates authorization restrictions, such as the enforcement of modal constraints, should not apply. This provides a mechanism for performing pure authentication operations. Unless the ACME$M_FOREIGN_POLICY_HINTS function modifier is also specified, this modifier applies only to the native domain of interpretation.

ACME$M_FOREIGN_POLICY_HINTS

The ACME$M_FOREIGN_POLICY_HINTS function modifier indicates ACME agents should honor the ACME$M_NOAUDIT and ACME$M_NOAUTHORIZATION function modifiers for non-native domains of interpretation.

ACME$M_OVERRIDE_MAPPING

The ACME$M_OVERRIDE_MAPPING function modifier allows for the acquisition of non-native credentials, during a persona merge or copy operation, when an externally authorized principal name maps to an OpenVMS username which differs from that associated with the native credentials. The default action is to disallow the commingling of credentials when a mapping conflict is detected; this modifier disables that action.

ACME$M_TIMEOUT

The ACME$M_TIMEOUT function modifier indicates the requested operation should time-out if no response is received within the time-out interval. The ACME$_TIMEOUT_INTERVAL item code can be used to specify the duration of the time-out period; thereby overriding the default interval.

ACME$M_UCS2

The ACME$M_UCS2 function modifier indicates item codes which specify string values use UCS-2 (UNICODE) representation rather than 8-bit ASCII.

3.1.6 Item Codes

This section describes the various item codes applicable to the function codes supported by the $ACM service

The item code space is partitioned into standard items and ACME-specific items. ACME-specific items are used to request information that is unique to a particular domain of interpretation. The item codes described in this section fall into the standard item code space.

Refer to section 3.1.4 for information about which item codes are applicable to each function.

ACME$_ACCESS_PORT

The ACME$_ACCESS_PORT item code is an input item code. It specifies the name of local device (terminal or access port) applicable to an authentication request. The buffer must contain a name string. If not specified, $ACM will pass the name string contained in the PCB$T_TERMINAL field of the process control block for the process.

ACME$_AUTH_MECHANISM

The ACME$_AUTH_MECHANISM item code is an input item code. It specifies the authentication mechanism applicable to an authentication request. The buffer must contain a word value specifying the desired mechanism code. If not specified, the authenticating domain of interpretation applies its default mechanism.

Standard mechanism types defined by the $ACMEDEF macro are:

ACME$K_CHALLENGE_RESPONSE

ACME$K_PASSWORD

ACME$_CHALLENGE_DATA

The ACME$_CHALLENGE_DATA item code is an input item code. It specifies the challenge data which was used as the basis for generating the response data specified the ACME$_RESPONSE_DATA item code. If this item code is specified, the ACME$_RESPONSE_DATA item code must also be specified.

ACME$_DIALOGUE_SUPPORT

The ACME$_DIALOGUE_SUPPORT item code is an input item code. It specifies which dialogue mode features which are supported by the caller. The buffer must contain a longword bit vector of applicable flags.

Valid flags are those defined for the ACMEIS$L_FLAGS field in an item set.

The ACME$_DIALOGUE_SUPPORT item code must be specified to use the interactive capabilities of $ACM. This item code must be the first entry in the item list. If the caller is unable to support features deemed necessary to complete a given request, the request ultimately fails.

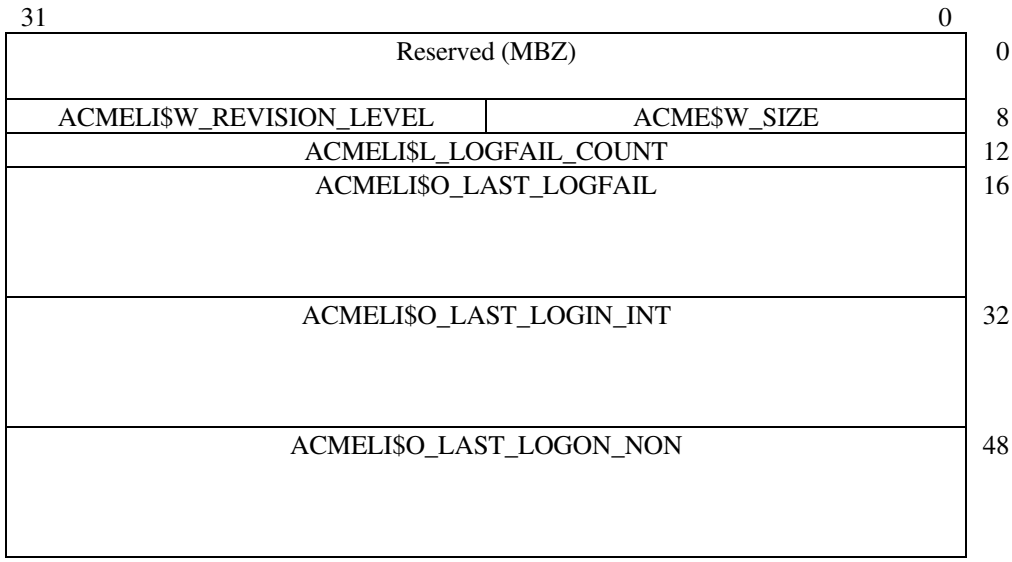
ACME$_EVENT_DATA

The ACME$_EVENT_DATA item code is an input item code. It specifies the buffer containing information applicable to an event operation.

ACME$_LOGON_INFORMATION

The ACME$_LOGON_INFORMATION item code is an output item code. It specifies the buffer to receive an ACM logon information structure which contains statistics pertaining to the authenticated principal name within the context of the authenticating domain of interpretation.. The size of the buffer must be equal to or greater than ACMELI$K_LENGTH.

The following diagram depicts the format of the ACM logon information structure.



The following table defines the ACM logon information structure fields

Descriptor Field	Definition
ACMELI$W_SIZE	A word containing the size of the size of the ACM logon information structure.

Descriptor Field	Definition
ACMELI$W_REVISION_LEVEL	A word containing a value which identifies the revision level of the ACM logon information structure.
ACMELI$L_LOGFAIL_COUNT	A longword containing the number of failed login attempts.
ACMELI$O_LOGFAIL	An octaword containing the date and time in UTC format of the last logon failure. (0 = undefined)
ACMELI$O_LAST_LOGON_INT	An octaword containing the date and time in UTC format of the last interactive logon. (0 = undefined)
ACMELI$O_LAST_LOGON_NON	An octaword containing the date and time in UTC format of the last non-interactive logon. (0 = undefined)

ACME$_LOGON_TYPE

The ACME$_LOGON_TYPE item code is an input item code. It specifies the type of logon being performed. The buffer must contain a longword value specifying a valid type.

The valid logon types defined by the $ACMEDEF macro are:

ACME$K_DETACHED

ACME$K_DIALUP

ACME$K_LOCAL

ACME$K_NETWORK

ACME$K_REMOTE

ACME$_MAPPED_VMS_USERNAME

The ACME$_MAPPED_VMS_USERNAME item code is an output item code. It specifies the buffer to receive the name of the local OpenVMS username to which the principal name was mapped. The buffer must be large enough to hold an ACME$S_MAX_VMS_USERNAME length name string.

ACME$_MAPPING_DOI

The ACME$_MAPPING_DOI item code is an output item code. It specifies the buffer to receive the name of the domain of interpretation which successfully mapped the principal name to an OpenVMS username. The buffer must be large enough to hold an ACME$S_MAX_DOI_NAME length name string.

ACME$_NEW_PASSWORD_1

The ACME$_NEW_PASSWORD_1 item code is an input item code. It specifies the new primary password for a password change operation. The buffer must contain a password string.

ACME$_NEW_PASSWORD_2

The ACME$_NEW_PASSWORD_2 item code is an input item code. It specifies the new secondary password for a password change operation. The buffer must contain a password string .

ACME$_NEW_PASSWORD_SYSTEM

The ACME$_NEW_PASSWORD_SYSTEM item code is an input item code. It specifies the new system password for a password change operation. The buffer must contain a password string .

ACME$_PASSWORD_1

The ACME$_PASSWORD_1 item code is an input item code. It specifies the primary password applicable to the requested operation. The buffer must specify a password string .

ACME$_PASSWORD_2

The ACME$_PASSWORD_2 item code is an input item code. It specifies the secondary password applicable to the requested operation. The buffer must specify a password string .

ACME$_PASSWORD_SYSTEM

The ACME$_PASSWORD_SYSTEM item code is an input item code. It specifies the system password applicable to the requested operation. The buffer must specify a password string .

ACME$_PERSONA_HANDLE_IN

The ACME$_PERSONA_HANDLE_IN item code is an input item code. It specifies the persona to use as the basis for credential acquisition processing. The buffer must contain a longword value specifying a valid persona-id..

ACME$_PERSONA_HANDLE_OUT

The ACME$_PERSONA_HANDLE_IN item code is an output item code. It specifies the buffer to receive the persona -id resulting form credential acquisition processing. The buffer descriptor must specify a longword..

ACME$_PRINCIPAL_NAME_IN

The ACME$_PRINCIPAL_NAME item code is an input item code. It specifies the name of the entity which is subject to authentication within the domain of interpretation to which it belongs. The buffer must contain a name string.

ACME$_PRINCIPAL_NAME_OUT

The ACME$_PRINCIPAL_NAME_OUT item code is an output item code. It specifies the buffer to receive the name of the entity which was authenticated by the authenticating domain of interpretation. This item code is useful when the principal name is not explicitly provided; such as during autologon style processing during which an ACME agent provides the principal name. The buffer must be large enough to hold an ACME$S_MAX_PRINCIPAL_NAME length name string..

ACME$_QUERY_DATA

The ACME$_QUERY_DATA item code is an output item code. It specifies the buffer to receive the data returned from the query operation relating to the most recent ACME$_QUERY_KEY item code.

ACME$_QUERY_DOI

The ACME$_QUERY_DOI is an input item code. It establishes the domain of interpretation within which query operations are performed. The buffer must contain the name of a domain of interpretation.

ACME$_QUERY_KEY

The ACME$_QUERY_KEY item code is an input item code. It specifies the key data for a query operation.

ACME$_QUERY_TYPE

The ACME$_QUERY_TYPE item code is an input item code. It specifies the key type for a query operation.

ACME$_REMOTE_HOST_ADDRESS

The ACME$_REMOTE_HOST_ADDRESS item code is an input item code. It specifies the network address of the system from which this request originated. The buffer must contain a network address using the representation consistent with ACME$_REMOTE_HOST_ADDRESS_TYPE. This item code is only meaningful only when the ACME$_REMOTE_HOST_ADDRESS_TYPE item code is specified.

ACME$_REMOTE_HOST_ADDRESS_TYPE

The ACME$_REMOTE_HOST_ADDRESS_TYPE item code is an input item code. It specifies the representation of the ACME$_REMOTE_HOST_ADDRESS. The buffer must contain a longword value specifying the address type.

Standard address types defined by the $ACMEDEF are:

ACME$K_DECNET_IV

ACME$K_DECNET_OSI

ACME$K_IP

ACME$_REMOTE_HOST_NAME

The ACME$_REMOTE_HOST_NAME item code is an input item code. It specifies the name of the remote system from which the request originated. The buffer must contain a name string.

ACME$_REMOTE_USER_NAME

The ACME$_REMOTE_USER_NAME item code is an input item code. It specifies the name of the remote user on whose behalf the request is being initiated. The buffer must contain a name string.

ACME$_RESPONSE_DATA

The ACME$_RESPONSE_DATA item code is an input item code. It specifies the response data which was calculated using the challenge data:

- provided by an ACME agent (as prompt data in the item set entry which requested the response data) during a dialogue mode authentication, or
- specified by the ACME$_CHALLENGE_DATA item code in a directed call to $ACM.

ACME$_TARGET_DOI

The ACME$_TARGET_DOI is an input item code. It establishes the domain of interpretation within which non-query operations are performed. The buffer must contain the name of a domain of interpretation.

ACME$_TIMEOUT_INTERVAL

The ACME$_TIMEOUT_INTERVAL item code is an input item code. It specifies the time-out interval in seconds. The buffer must contain a longword value. If omitted or specified as 0, the default time-out value is used. This item code is ignored when the ACME$M_TIMEOUT function modifier is not specified.

Special Item Codes

ACME$_CHAIN

The ACME$_CHAIN item code is an input item code. It specifies the address of another item list to processing immediately after processing the current list. The buffer address field in the item descriptor specifies the address of the next item list to be processed. The ACME$_CHAIN item code must be last

in the item list; $ACM treats this as the logical end of the current item list. All item list entries following ACME$_CHAIN item code are ignored.

On the Alpha platform both 32-bit and 64-bit item lists can be chained together.

ACME$_IGNORE
ACME$_NULL
ACME$_NULL_AND_VOID
ACME$_NULL_AND_AVOID
ACME$_PLACEHOLDER
ACME$_SKIP
ACME$_VOID

The ACME$_IGNORE item code indicates the current item list entry is to be ignored.

ACME$_NULL, ACME$_NULL_AND_VOID, ACME$_NULL_AND_AVOID, ACME$_PLACEHOLDER, ACME$_SKIP and ACME$_VOID are all synonyms for ACME$_IGNORE.

3.1.7 Description

The Authentication and Credential Management ($ACM) service presents a unified interface for performing authentication related operations in a manner independent of applicable policy.

On a given OpenVMS system, multiple authentication policies may be applicable. The system may be configured to augment the native (local OpenVMS) policy with alternatives pertaining to external environments, such as LAN Manager. Each policy, together with the operating environment to which it pertains, constitutes a ***domain of interpretation***. Within a given domain, any entity, such as a user, which is subject to the applicable authentication policy, is referred to as a ***principal***.

The $ACM service can be used to authenticate a principal, initiate a password change request on behalf of a principal, inquire about a principal or the operating environment, or report event data within a particular domain.

The $ACM service completes asynchronously; that is, it returns to the caller after queuing the request, without waiting for the operation to complete.

To synchronize completion of an operation, use the Authentication and Credential Management and Wait ($ACMW) service. The $ACMW service is identical to $ACM in every way except that $ACMW returns to the caller after the operation has completed.

Modes of Operation

The typical authentication policy employs the traditional reusable password; however, various alternative mechanisms exist for forming stronger policies. Some of these mechanisms, such as challenge-response, require interaction. The $ACM service is designed to accommodate these mechanisms.

The ***authentication*** and ***change password*** functions are capable of operating in a dialogue (iterative) mode to support different types of interactive authentication mechanisms. The ***query***, ***event***, and ***free context*** functions only support the non-dialogue (non-iterative) mode of operation..

Non-dialogue (Non-iterative) Mode

The default nature of the $ACM service is to operate in a non-iterative mode. All information needed to complete the request must be provided in a single call ; otherwise, the request ultimately fails In effect, this requires the caller to know beforehand what information is required to complete the request.

The following list summarizes the control flow for a typical non-dialogue mode authentication request. For simplicity, the scenario assumes a single domain of interpretation with a traditional username + password based policy. Also, error processing is ignored.

1. Caller of $ACM prompts for the principal name and password, builds an item list specifying the principal name and password, and calls $ACM specifying the authenticate principal function, the item list with the principal name and password, and a context argument of 0.
2. $ACM processes the request and ultimately returns control to the caller with the final status for the operation.

Dialogue (Iterative) Mode

The caller can use the interactive capabilities of the $ACM service for authentication and password change operations by specifying the ACME$_DIALOGUE_SUPPORT item code and a valid **context** argument. In this mode, ACME agents can request additional information from the caller in order to complete the request. In effect, the $ACM service is called in an iterative fashion until all information required to complete the request has been provided. The sequence of calls are linked together by passing the **context** argument returned in one call back in the next call.

In this scenario, when an ACME agent requires additional information, it builds an item set list which describes the nature of the information . The item set list is passed back to the caller in the communications buffer (refer to the description for the **context** argument regarding the format of the communications buffer) and the service returns with the ACME$_OPINCOMPL status. The caller processes each item set entry, gathers the requested information, and then passes it back to the ACME agent using the **itmlst** argument in the next call. The sequence continues until the call returns with a status code other than ACME$_OPINCOMPL.

The following list summarizes the control flow for a typical dialogue mode authentication sequence. For simplicity, the scenario assumes a single domain of interpretation with a traditional username+password based policy. Also, error processing is ignored.

1. Make an initial call to $ACM specifying the authenticate principal function code, an item list that merely contains the ACME$_DIALOGUE_SUPPORT item code, and a context argument which has been initialized to 0.
2. $ACM builds a communications buffer containing an item set list in the buffer requesting the principal name (username), sets the context argument to reference the buffer, and returns control to the caller with a status code of ACME$_OPINCOMPL.
3. The caller processes the item set list, prompts for the principal name, builds an item list specifying the principal name, and calls $ACM again specifying the authenticate principal function as before, the item list with the principal name, and a context argument which contains the buffer address returned in the previous call.
4. $ACM validates the context argument and clears it, rebuilds the communications buffer to contain an item set list requesting the password, sets the context argument to reference the buffer; and returns control to the caller once again with a status code of ACME$_OPINCOMPL.
5. The caller processes the item set list, prompts for the password; builds an item list specifying the password, and calls $ACM again specifying the authenticate principal functions as before, the item list with the password, and a context argument which contains the buffer address returned in the previous call.
6. $ACM validates the context argument again, clears it, and then completes processing of the request, now that it has all the necessary information, and ultimately returns control to the caller with the final status for the operation.

Privileges and Restrictions

The $ACM service constitutes a trusted interface. It restricts operations which override the security policy applicable to a given domain of interpretation to those callers who are sufficiently privileged. In general, attempts to breach the security policy of a domain of interpretation by a caller who is not suitably privileged for the requested operation fail in a generic fashion. The failure status returned in these situations is non-specific, so as to not inadvertently reveal sensitive information.

To specify the following function codes the caller must have the SECURITY privilege.

ACME$_FC_SHUTDOWN

ACME$_FC_STARTUP

To specify the following function modifiers the caller must have the SECURITY privilege.

ACME$M_NATIVE_AUTHENTICATION

ACME$M_NOAUDIT

ACME$M_NOAUTHORIZATION

ACME$M_FOREIGN_POLICY_HINTS

To specify the following function modifier the caller must have the IMPERSONATE privilege.

ACME$M_OVERRIDE_MAPPING

To specify the following item code the caller must have both the SECURITY and CMKRNL privileges.

ACME$_NEW_PASSWORD_SYSTEM

To specify the following item codes the caller must have the IMPERSONATE privilege

ACME$_ACCESS_PORT

ACME$_CHALLENGE_DATA

ACME$_LOGON_TYPE

ACME$_REMOTE_HOST_ADDRESS

ACME$_REMOTE_HOST_ADDRESS_TYPE

ACME$_REMOTE_HOST_NAME

ACME$_REMOTE_USER_NAME

3.1.8 Condition Values Returned in R0

SS$_NORMAL	The service completed successfully.
SS$_ACCVIO	The item list or an input buffer cannot be read in the access mode of the caller; or an output buffer, a return length buffer, or the I/O status block cannot be written in the access mode of the caller.
SS$_ARG_GTR_32_BITS	A 64-bit address was passed in a context requiring a 32-bit address.
SS$_BADBUFADR	The buffer address associated with an entry in the item list is inappropriate in the context of the call. The address may be invalid (for example, 0).
SS$_BADBUFLEN	The buffer length associated with an entry in the item list is inappropriate in the context of the call. The length may be invalid (for example, 0) or outside the range of acceptable values.

SS$_BADCHAIN	A chained item list is inaccessible or the chain is circular.
SS$_BADCONTEXT	The **context** argument does not specify a valid context buffer.
SS$_BADITMCOD	A specified item code is invalid or out-of-range.
SS$_BADPARAM	The item list contains an invalid item code.
SS$_BADRETLEN	The return length address associated with an entry in the item list is inappropriate in the context of the call. The address may be invalid (for example, 0).
SS$_EXASTLM	The **astadr** argument was specified and the process has exceeded its ASTLM quota.
SS$_EXQUOTA	A process quota was exceeded.
SS$_ILLEFC	The **efn** argument specifies an illegal event flag number.
SS$_ILLMODIFIER	The **func** argument specifies function modifiers which are inappropriate in the context of the call.
SS$_INSFMEM	Insufficient space exists for completing the request.
SS$_NOEXTAUTH	External authentication is not available.
SS$_NOPRIV	The caller does not have the necessary privileges to complete the requested operation.
SS$_UNASEFC	The **efn** argument specifies an unassociated event flag cluster.
SS$_UNSUPPORTED	The **func** argument specifies an unsupported function.

3.1.9 Condition Values Returned in the I/O Status Block

ACME$_NORMAL	The service completed successfully.
ACME$_ACCOUNTLOCK	The account associated with specified principal name is disabled.
ACME$_AUTHFAILURE	Authorization failed..
ACME$_BUFFEROVR	An output item returned the service is larger than the user buffer provided to receive the item; the item is truncated.
ACME$_DOIUNAVAILABLE	The specified domain of interpretation is not processing requests.
ACME$_INTRUDER	A record matching the request was found in the intrusion database.
ACME$_INVALIDCTX	The **context** argument specified is invalid.
ACME$_INVALIDPWD	The specified password is invalid.
ACME$_INVPERSONA	The persona handle specified by the **itmlst** argument is invalid.
ACME$_INVREQUEST	A parameter is invalid in the context of the request.
ACME$_MAPCONFLICT	An attempt was made to merge credentials for a principal name which map to a OpenVMS username that differs from that associated with existing credentials..
ACME$_NOTAUTH	Authorization failed due to account restrictions.
ACME$_NOEXTAUTH	The specified principal name can not be authenticated externally.
ACME$_NOPRIV	The caller does not have the necessary privileges to complete the

	requested operation.
ACME$_NOSUCHDOI	The specified domain of interpretation does not exist.
ACME$_NOSUCHUSER	The specified principal name does not exist.
ACME$_NOTARGETCREDS	The persona does not contain credentials for the specified domain of interpretation.
ACME$_OPINCOMPL	Interaction is required in order to complete the request. The context buffer contains information describing how to proceed.
ACME$_TIMEOUT	The server did not respond within the designated time-out interval.
ACME$_UNSUPPORTED	The requested operation or an item code is not supported with the selected domain of interpretation.

3.2 ACME Agent Requirements

The ACME agent is responsible for enforcing policy according to the requirements of its domain of interpretation and for properly interacting with $ACM callers in dialogue mode and coordinating its activities with the security server. The ACME agent can also provide its own error logging mechanism as an aid in diagnosing problems specific to its DOI.

Details on these requirements and the actual interface will be described in the SYS$ACM design specification.

4. OpenVMS SSPI Operation

The Microsoft-compatible Security Support Provider Interface (SSPI) for NTLM distributed security is being provided as a separate component of the Single Sign-On infrastructure project. The OpenVMS SSPI (and NTLM-SSP), itself a system service, will use the ACM authority for authentication services in a manner similar to how the Microsoft SSPI and NTLM-SSP would use the Windows NT Local Security Authority.

The SSPI will call $ACM using the non-dialogue mode of operation to authenticate an NT principal (see section 8 for further details):

ACME$_TARGET_DOI	“NT_LM”
ACME$_PRINCIPAL_NAME	“joe@domain”
ACME$_AUTH_MECHANISM	ACME$K_CHALLENGE_RESPONSE
ACME$_CHALLENGE	challenge-data
ACME$_RESPONSE	response data

The NT_LM ACME agent supplies “workstation” field for network logon type (SCSNODE name).

5. Impact on Existing Base Components

LOGINOUT and the SET PASSWORD command will be revised to utilize the $ACM service. In addition, the SYS$SINGLE_SIGNON bit-mask controls from OpenVMS version 7.1 will retain their usage with the exceptions noted below.

This section briefly describes what work needs to be done for each.

5.1 SYS$SINGLE_SIGNON Bit-Mask Controls

Bit 0 (ext. auth.)	No longer supported. Replaced by ACME agent START/STOP feature.
Bit 1 (local auth.)	Supported only by LOGINOUT to allow non-privileged use of the /LOCAL_PASSWORD login qualifier.

5.2 LOGINOUT

1. The native ACME agent for OpenVMS will support the username + password-based authentication policy currently resident in LOGINOUT. LOGINOUT will rely on the $ACM service for policy enforcement.
2. LOGINOUT will use dialogue-mode for interactive logons and non-dialogue mode for non-interactive logon operations.
3. LOGINOUT will use the ACME$M_NATIVE_AUTHENTICATION function modifier to force native authentication when the /LOCAL_PASSWORD qualifier is specified.
4. The implicit application of the /LOCAL_PASSWORD qualifier, when SYS$SINGLE_SIGNON bit 1 is set, will no longer be supported. Explicit specification of the /LOCAL_PASSWORD qualifier will be required. Bit 1 will only be used to allow the non-privileged use of the /LOCAL_PASSWORD qualifier.
5. A mechanism will be provided which preserves the functionality of the current LGI callout interface. If LGI callouts are installed, they will continue to operate as before.
6. If the ACM authority is unavailable, LOGINOUT will provide for an emergency login into the SYSTEM account via the system console (OPA0:). This-strategy parallels the situation in which the SYSUAF file is unavailable.
7. SYSUAF password synchronization will still occur during logon (LOGINOUT). This is required until modifications to the DECwindows product are updated to use the $ACM service for unlocking the pause screen.

5.3 SET PASSWORD

1. The native ACME agent for OpenVMS will support the password-change policy currently resident in the SET PASSWORD utility. The SET PASSWORD utility will rely on the $ACM service for policy enforcement.
2. The SET PASSWORD utility will use dialogue-mode for password change operations.
3. The SET PASSWORD utility will use the principal identified by the primary credentials contained

in the currently active persona as the default basis for password change operations. If the active persona does not contain a set of primary credentials, the SET PASSWORD utility will prompt for the principal name and perform a change password operation within the context of the domain of interpretation which maps the principal.

4. A mechanism will be provided which preserves the functionality of the current site specific password filtering interface. If site specific password filters are installed, they will continue to operate as before.
5. If ACM authority is unavailable, password change operations via the SET PASSWORD utility will be disabled.

5.4 Authorize and $SETUAI/$GETUAI

1. A new UAF$V_DISPWDSYNCH flag is needed to disable local SYSUAF password synchronization. This arises from the situation in which multiple principals map to the same local SYSUAF account. In this situation, password synchronization would cause a password change for one principal to overwrite the password change initiated by another

6. Issues

6.1 Open Issues

1. Should there be a switch for LOGINOUT and SET PASSWORD to fall back to previous behavior. That is, work directly against the SYSUAF.DAT rather than be forced to follow the $ACM path through the security server? (This is how external authentication is implemented in version 7.1, based on the SYS$SINGLE_SIGNON bit-mask settings.)

   (Internal review felt that all authentication should now pass through this interface.)

2. Is second granularity fine enough for time-outs?

The following key issues were raised during the external review of X 0.2 of this document

3. The needs of the Events/Auditing are not compeletely defined.
4. Where will the ACME interface details be publicly documented.

   For Raven, tt has not been decided if we will publicly document this interface as part of the standard OpenVMS documentation. We will make it available to our partners as needed.

6.2 Resolved Issues

1. **What do we call this system service?**
   We will use the name SYS$ACM (Authentication and Credential Management) Service.
2. **How do we provide a flexible dialog for different password schemes?**
   The iterative dialog solves this problem.
3. **Is SYS$ACM an exec or kernel mode call?**
   Kernel mode for access from all caller modes, in particular the kernel-mode SSPI.
4. **How to input/display UNICODE to a 8-bit character device?**
   The service interface supports both 8-bit ASCII and UCS-2. Clients (such as LOGINOUT) will use the appropriate representation for a given locale and will handle any mapping requirements relating to input/output.
5. **Who will provide UNICODE utility routines?**
   UNICODE project. Project leader: Chris Chan.

The following key issues were raised during the external review of X 0.2 of this document.

6. **Description of service parameters and usage inadequate, information is scattered.**
   The document has been restructured to consolidate related information. Details surrounding the service interface have been expanded and now use a format similar to that in the OpenVMS System Services Reference manuals.
7. **Level of privileges needed for each operation needs to be reviewed/clarified.**
   The strategy surrounding privileges has been revised to require privilege only for operations which would circumvent policy. Refer to the **Privileges and Restrictions** section of the service description for more detail.

8. **Clarify UNICODE strategy.**
   The service interface supports both 8-bit ASCII (default) and UCS-2. The ACME$M_UCS2 function modifier has been introduced to specify which representation the caller is using.

9. **Prompting for username, password, etc. should be driven by authentication policy.**
   The dialogue mode of operation provides a mechanism whereby authentication activities can be directed by the ACME agent(s)

10. **Should the service be allocating the communications buffer?**
    A variety of services allocate buffers on behalf on the caller. Typically, such buffers are used internally by the service and are not directly visible to the user. In contrast, the dialogue mode of operation requires a mechanism for communicating prompt information, etc. to the caller. Having the service allocate the buffer ensures it will be of sufficient size and avoids the overhead of multiple calls needed to ensure so otherwise. Since $ACM can not assume the caller is 64-bit aware, the buffer must be allocated withn the 32-bit address space. As for other services, the necessary hooks will be put in place to ensure cleanup during image/process rundown.

11. **Usage of SYS$SINGLE_SIGNON logical to control operations?**
    Usage of the SYS$SINGLE_SIGNON logicial to control ACME behavior will be replaced by an alternative mechanism, the details of which do not affect the general aspects of the $ACM interface and will be addressed in the SYS$ACM Design Specification.

12. **How to control the ACME ordering?**
    The sequence in which ACME agents are called will be controlled by a dynamically configurable mechanism, the details of which do not affect the general aspects of the $ACM interface and will be addressed by the SYS$ACM Design Specification.

13. **Can ACME ordering change between iterative calls?**
    No. ACME ordering can only be changed while no requests are outstanding.

14. **Can ACME ordering change between logins?**
    Yes, provided there are no outstanding requests.

15. **In what order are item in the information buffer processed?**
    Item set entries in the communcitiona buffer are processed sequentially in the order in which they appear in the buffer. The description of the **context** argument has been expanded to call this out.

16. **The service should take 64-bit guidelines into consideration.**
    This has always been the intent. The service description has been expanded to include details clarifying this point.

17. **Is the EFN argument needed?**
    Stu Davidson points out that threading models and AST support address different programming paradigms. The **efn**, **astadr**, and **astprm** arguments relate to supporting the AST programming model. If this service was to be purely thread oriented, dropping these parameters would be the right thing to do. But as Stu points out, switching programming paradigms truly need to be addressed at a broader level and is beyond the scope of this project.

18. **What level of support for credentials is provided on the VAX platform?**
    Credential support on the VAX plarform will be limited to that which is currently provided by the existing impersonation services.

19. **PATHWORKS redirector needs call $ACM with a challenge and receive back both the NT and LM responses.**
    This will be handled through an NT_LM ACME specific query operation.

20. **Clarify impact on LGI callouts.**
    The description of LOGINOUT changes has been updated to clarify this issue.

21. **Clarify impact on DECwindows.**
    DECwindows login and pause screen components should be integrated to use the $ACM service. This issue has been brought to the attention of DECwindows product management and its resolution resides solely within their realm.

7. Interactive Authentication Prototypes

For Raven, it is mandatory that both native OpenVMS authentication (primary and optional secondary password) as well as LAN Manager authentication (single password) are supported. However, this interface is designed to allow other authentication mechanisms to replace the traditional reusable password mechanism.

This section contains examples of three popular authentication products that replace or augment a reusable password policy: In each case, the prompting sequence and data types are described. These were used as the prototype mechanisms that the interface had to support for Raven.

- **SecurID Token** (Security Dynamics Technologies, Inc.)
- **CRYPTOCard RB-1 Token** (CRYPTOCard Corporation)
- **S/Key One-Time Password System** (Bell Communications Research, Inc.)

Each of these prototypes can use the ACME$K_PASSWORD mechanism type. Prototype-specific item codes can be used to process any non-password information.

7.1 SecurID Token

The SecurID token from Security Dynamics is a hardware device about the size of a credit card. It provides two-factor authentication (something one knows and something one possesses). The token displays an unpredictable one-time access code every 60 seconds. Access code generation is synchronized with an access control module. If the access code and PIN entered by the user matches the one in the access control module, the user is validated. There are two flavors of the SecurID token: a) clear text passcode, and b) encrypted passcode (using a token with a keypad for entering the user's PIN). The prompting sequence is identical for both token types, but the value of the passcode that is entered is different.

The sequence for the clear text passcode token (input shown in bold type):

Username: **joesmith**
Passcode: **12399999999** (PIN and access code)

[OpenVMS password can also be required, depending on user settings]

(access granted or denied)

The sequence for the encrypted passcode token (input shown in bold type):

Username: **joesmith**
Passcode: **88888888** (encrypted PIN and access code)

[*OpenVMS password can also be required, depending on user settings*]

(access granted or denied)

7.2 CRYPTOCard RB-1 Token

CRYPTOCard Corporation's CRYPTOCard RB-1 token is another two-factor authentication device which effectively provides one-time password authentication. Also about the size of a credit card, the RB-1 token uses a DES-based encryption algorithm to compute and respond to a challenge issued by the

authenticating system. The user enters his or her PIN into the token along with the challenge displayed by the system. The token uses the PIN to calculate the response to the challenge and displays the result. The user then enters this result into the system to complete the authentication sequence. There are two flavors of operation: a) the method just described, and b) event-synchronous mode which operates in a similar fashion but the token will display the next expected challenge in an effort to reduce keystrokes. The prompting sequence for both types of operation are identical as far the authenticating system is concerned.

The sequence is as follows (input shown in bold type):

Username: **joesmith**
RB-1 challenge is 98765432
Response: **A120BCDF**

[*OpenVMS password can also be required, depending on user settings*]

(access granted or denied)

7.3 S/Key One-Time Password System

The S/Key one-time password system from Bellcore is described in Internet Engineering Task Force RFC 1938. S/Key provides one-time password operation based on a list of one-time passwords generated from a secret pass phrase, passing the initial pass phrase through a one-way hash function and then repeating the operation a specified number of times using the result of the previous iteration as input. The list and the number of passwords are stored in a secure database. Each time authentication is required, the password number is reduced by one. To perform authentication, the system issues a random challenge to the user who must compute the correct response using some other software or hardware component and enter it into the system to complete the sequence.

The sequence is as follows (input shown in bold type):

Username: **joesmith**
otp-md5 487 dog2
S/Key password: **oust coat foal mug beak tote**

[*OpenVMS password can also be required, depending on user settings*]

(access granted or denied)

8. Windows NT-Specific Information

This section contains information specific to the Windows NT environment as it relates to the MSV1_0 authentication model used by NT up through version 4.0.

8.1 MSV1_0 Credentials

Windows NT principals possess a set of security attributes that constitute its *primary* credentials. These security attributes are obtained from a domain controller upon successful authentication (via PATHWORKS in this case). In Windows 4.0 and earlier, the authentication model is based strictly on passwords and is referred to as MSV1_0 authentication.

MSV1_0 security attributes that will represent the user's NT credentials as a persona extension block are the following (see *Windows NT Local Security Authority - Authentication*, pg. 36):

DomainName *Domain name in UCS-2*
UserName *User account name in UCS-2*
NtOwfPassword *One-way function of the user's NT password (UCS-2, case-sensitive)*
LmOwfPassword *One-way function of the user's LM password (8-bit, case-insensitive)*
NtPasswordPresent *Flag indicating the presence of the NT OWF password*
LmPasswordPresent *Flag indicating the presence of the LM OWF password*
NtAccessToken *Access token upon which access checks are made. The access token is composed of the following fields:*

- *user security identifier (SID)*
- *group SIDs*
- *primary SID*
- *privileges*
- *owner SID*
- *default access control list*

8.2 MSV1_0 Logon Input/Output Parameters

The input parameters required for an **interactive logon** are (see *Windows NT Local Security Authority - Authentication*, pg. 37):

DomainName *Domain name in UCS-2*
UserName *User account name in UCS-2*
Password *Clear text password in UCS-2*

Interactive logons return the following user account information of interest:

BadPasswordCount *Number of invalid password attempts since last successful logon*
LogonTime *Time of last successful logon in GMT*
PasswordCanChange *Time when the user can change the password (may be never) in GMT*
PasswordMustChange *Time when the user must change the password (may be never) in GMT*

The input parameters required for a **network logon** are:

DomainName *Domain name in UCS-2*
UserName *User account name in UCS-2*
WorkStation *Computer name of client workstation in UCS-2*
ChallengeToClient *Original value received from server*

CaseSensitiveResponse	*Challenge response based on the client's NT OWF password and server challenge (may be null string)*
CaseInsensitiveResponse	*Challenge response based on the client's LM OWF password and server challenge (may be null string)*

Network logons return the following user account information of interest:

UserSessionKey	*Encryption key based on the NT OWF password*
LmSessionKey	*Encryption key based on the LM OWF password*

These two types of logon will be implemented using the challenge-response mechanism at the NT_LM ACME agent-PATHWORKS server interface. The $ACM service interface will present both password and challenge-response mechanisms.

8.3 NT_LM ACME Constants and Item Codes

This section contains constants and item codes specific to the NT_LM ACME agent.

The following constants are used for SID-to-AccountName and PrivilegeLUID-to-PrivilegeName mappings using the $ACM service ACME$_FC_QUERY_DATA function. These constants define the key types associated with the ACME$_QUERY_TYPE item code.

NTA$_ACCOUNT_NAME_KEY

NTA$_CHALLENGE_KEY

NTA$_PRIVILEGE_LUID_KEY

NTA$_PRIVILEGE_NAME_KEY

NTA$_SID_KEY

The following item codes are used for the ACME$_FC_EVENT function:

NTA$_ACCESS_AUDIT

9. Glossary

ACM	Authentication and Credential Management.
ACME	Authentication and Credential Management Extension.
Credentials	Information used to prove the identity of a principal or for authorization purposes.
DOI	Domain of interpretation.
One-Time Password	A mechanism used for authentication where the password changes after each use to protect against stolen passwords or sharing of passwords.
Persona	OpenVMS internal data structure containing a principal's credentials.
Persona Extension	Additional data structure attached to a persona that contains non-native information about the principal.
Primary Credentials	Credentials obtained from a DOI which resulted in the creation of the original persona. Set by the $ACM service typically during LOGINOUT. Used by certain components to determine the "logon" identity of the principal, such as SET PASSWORD.
Principal	User (human) or entity operating on behalf of a user.
UCS-2	Standard two-byte UNICODE.