Security Framework Reference

Security > Carbon

2008-03-12

Apple Inc.
© 2008 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, AppleTalk, Keychain, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

iPhone is a trademark of Apple Inc.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Introduction

Framework	/System/Library/Frameworks/Security.framework
Header file directories	/System/Library/Frameworks/Security.framework/Headers
Companion guide	Secure Coding Guide
Declared in	Authorization.h AuthorizationDB.h AuthorizationTags.h CipherSuite.h SecACL.h SecAccess.h SecBase.h SecCertificate.h SecIdentity.h SecIdentitySearch.h SecImportExport.h SecKey.h SecKeychain.h SecKeychainItem.h SecKeychainSearch.h SecPolicy.h SecPolicySearch.h SecTrust.h SecTrustSettings.h SecTrustedApplication.h SecureTransport.h cssmapple.h cssmtype.h

This collection of documents provides the API reference for the Security framework, which defines C interfaces for protecting information and controlling access to software.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Introduction

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Managers

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

PART I

Managers

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Authorization Services C Reference

Framework: Security

Declared in Authorization.h
AuthorizationDB.h
AuthorizationTags.h

Overview

Authorization Services is an API that facilitates access control to restricted areas of the operating system and allows you to restrict a user's access to particular features in your Mac OS X application. Authorization Services is used in

Note: This document was previously titled *Authorization Services Reference*.

- software that restricts access to its own tools
- applications that call system tools
- software installers that install privileged tools or require access to restricted areas of the operating system

A companion volume to *Authorization Services C Reference* is Performing Privileged Operations With Authorization Services, which explains the concepts behind authorization and provides examples of how to use Authorization Services. Objective-C methods that are equivalent to several of the functions in this document are described in Authorization Services Objective-C Reference.

Authorization Services is available in Mac OS X v10.0 and later as part of the Security framework.

Functions by Task

Creating and Releasing Authorization References

`AuthorizationCreate` (page 14)
Creates a new authorization reference and provides an option to authorize or preauthorize rights.

`AuthorizationFree` (page 17)
Frees the memory associated with an authorization reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Requesting Rights and Credentials

`AuthorizationCopyRights` (page 12)
Authorizes and preauthorizes rights.

`AuthorizationCopyInfo` (page 10)
Retrieves side-band data such as the user name and other information gathered during evaluation of authorization.

`AuthorizationFreeItemSet` (page 18)
Frees the memory associated with an authorization set.

Externalizing and Internalizing Authorization References

`AuthorizationMakeExternalForm` (page 18)
Creates an external representation of an authorization reference.

`AuthorizationCreateFromExternalForm` (page 15)
Internalizes the external representation of an authorization reference.

Modifying the Policy Database

`AuthorizationRightGet` (page 19)
Retrieves a right definition as a dictionary.

`AuthorizationRightSet` (page 20)
Creates or updates a right entry in the policy database.

`AuthorizationRightRemove` (page 20)
Removes a right from the policy database.

Executing With Root Privileges

`AuthorizationExecuteWithPrivileges` (page 16)
Runs an executable tool with root privileges.

`AuthorizationCopyPrivilegedReference` (page 11)
Retrieves the authorization reference passed by the `AuthorizationExecuteWithPrivileges` function.

Functions

AuthorizationCopyInfo

Retrieves side-band data such as the user name and other information gathered during evaluation of authorization.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus AuthorizationCopyInfo (
   AuthorizationRef authorization,
   AuthorizationString tag,
   AuthorizationItemSet **info
);
```

Parameters

authorization

An authorization reference referring to the authorization session.

tag

An authorization string specifying the type of data the Security Server should return. Pass `NULL` to retrieve all available information.

info

A pointer to an authorization set the Security Server creates. On return, this set contains side-band authorization data. When this set is no longer needed, free the memory associated with it by calling the function `AuthorizationFreeItemSet` (page 18).

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
An authorization plug-in can store the results of an authentication operation by calling the `SetContextValue` function. You can use the `AuthorizationCopyInfo` function to retrieve this information.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationCopyPrivilegedReference

Retrieves the authorization reference passed by the `AuthorizationExecuteWithPrivileges` function.

```
OSStatus AuthorizationCopyPrivilegedReference (
   AuthorizationRef *authorization,
   AuthorizationFlags flags
);
```

Parameters

authorization

A pointer to an authorization reference. The Security Server allocates the authorization reference for you, so you do not need to call the function `AuthorizationCreate` (page 14). On return, it points to a copy of the authorization reference used in the call to the `AuthorizationExecuteWithPrivileges` (page 16) function.

flags

Reserved options. Pass the `kAuthorizationFlagDefaults` constant.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
This function retrieves the authorization reference you pass in the function `AuthorizationExecuteWithPrivileges` (page 16). The new process can use the authorization reference to verify authorizations obtained by the calling process.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
MoreIsBetter
QISA

Declared In
`Authorization.h`

AuthorizationCopyRights

Authorizes and preauthorizes rights.

```
OSStatus AuthorizationCopyRights (
   AuthorizationRef authorization,
   const AuthorizationRights *rights,
   const AuthorizationEnvironment *environment,
   AuthorizationFlags flags,
   AuthorizationRights **authorizedRights
);
```

Parameters

authorization
An authorization reference referring to the authorization session.

rights
A pointer to a set of authorization rights you create. Pass `NULL` if the application requires no rights at this time.

environment
Data used when authorizing or preauthorizing rights. Not used in Mac OS X v10.2 and earlier. In Mac OS X v10.3 and later, you can pass icon or prompt data to be used in the authentication dialog box. In Mac OS X v10.4 and later, you can also pass a user name and password in order to authorize a user without displaying the authentication dialog box. Possible values for this parameter are listed in `Security.framework/Headers/AuthorizationTags.h`. The data passed in this parameter is not stored in the authorization reference; it is used only during authorization. If you are not passing any data in this parameter, pass the constant `kAuthorizationEmptyEnvironment`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

flags

A bit mask for specifying authorization options. Use the following option sets.

- Pass the constant `kAuthorizationFlagDefaults` if no options are necessary.
- Specify the `kAuthorizationFlagExtendRights` mask to request rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPartialRights` and `kAuthorizationFlagExtendRights` masks to request partial rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPreAuthorize` and `kAuthorizationFlagExtendRights` masks to preauthorize rights.
- Specify the `kAuthorizationFlagDestroyRights` mask to prevent the Security Server from preserving the rights obtained during this call.

authorizedRights

A pointer to a newly allocated `AuthorizationRights` structure. On return, this structure contains the rights granted by the Security framework. If you do not require this information, pass `NULL`. If you specify the `kAuthorizationFlagPreAuthorize` mask in the `flags` parameter, the method returns all the requested rights, including those not granted, but the flags of the rights that could not be preauthorized include the `kAuthorizationFlagCanNotPreAuthorize` bit.

Free the memory associated with this set by calling the function `AuthorizationFreeItemSet` (page 18).

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
There are three main reasons to use this function. The first reason is to preauthorize rights by specifying the `kAuthorizationFlagPreAuthorize`, `kAuthorizationFlagInteractionAllowed`, and `kAuthorizationFlagExtendRights` masks as authorization options. Preauthorization is most useful when a right has a zero timeout. For example, you can preauthorize in the application and if it succeeds, call the helper tool and request authorization. This eliminates calling the helper tool if the Security Server cannot later authorize the specified rights.

The second reason to use this function is to authorize rights before performing a privileged operation by specifying the `kAuthorizationFlagInteractionAllowed`, and `kAuthorizationFlagExtendRights` masks as authorization options.

The third reason to use this function is to authorize partial rights. By specifying the `kAuthorizationFlagPartialRights`, `kAuthorizationFlagInteractionAllowed`, and `kAuthorizationFlagExtendRights` masks as authorization options, the Security Server grants all rights it can authorize. On return, the authorized set contains all the rights.

If you do not specify the `kAuthorizationFlagPartialRights` mask and the Security Server denies at least one right, then the status of this function on return is `errAuthorizationDenied`.

If you do not specify the `kAuthorizationFlagInteractionAllowed` mask and the Security Server requires user interaction, then the status of this function on return is `errAuthorizationInteractionNotAllowed`.

If you specify the `kAuthorizationFlagInteractionAllowed` mask and the user cancels the authentication process, then the status of this function on return is `errAuthorizationCanceled`.

Availability
Available in Mac OS X v10.0 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Related Sample Code
AuthForAll
BSDLLCTest
MoreIsBetter

Declared In
`Authorization.h`

AuthorizationCreate

Creates a new authorization reference and provides an option to authorize or preauthorize rights.

```
OSStatus AuthorizationCreate (
   const AuthorizationRights *rights,
   const AuthorizationEnvironment *environment,
   AuthorizationFlags flags,
   AuthorizationRef *authorization
);
```

Parameters

rights

A pointer to a set of authorization rights you create. Pass `NULL` if the application requires no rights at this time.

environment

Data used when authorizing or preauthorizing rights. Not used in Mac OS X v10.2 and earlier. In Mac OS X v10.3 and later, you can pass icon or prompt data to be used in the authentication dialog box. In Mac OS X v10.4 and later, you can also pass a user name and password in order to authorize a user without user interaction. Possible values for this parameter are listed in `Security.framework/Headers/AuthorizationTags.h`. The data passed in this parameter is not stored in the authorization reference; it is used only during authorization. If you are not passing any data in this parameter, pass the constant `kAuthorizationEmptyEnvironment`.

flags

A bit mask for specifying authorization options. Use the following option sets.

- Pass the constant `kAuthorizationFlagDefaults` if no options are necessary.
- Specify the `kAuthorizationFlagExtendRights` mask to request rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPartialRights` and `kAuthorizationFlagExtendRights` masks to request partial rights. You can also specify the `kAuthorizationFlagInteractionAllowed` mask to allow user interaction.
- Specify the `kAuthorizationFlagPreAuthorize` and `kAuthorizationFlagExtendRights` masks to preauthorize rights.
- Specify the `kAuthorizationFlagDestroyRights` mask to prevent the Security Server from preserving the rights obtained during this call.

authorization

A pointer to an authorization reference. On return, this parameter refers to the authorization session the Security Server creates. Pass `NULL` if you require a function result but no authorization reference.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
The primary purpose of this function is to create the opaque authorization reference structure associated with the authorization reference. You use the authorization reference in other authorization functions.

You can use this function to authorize all or partial rights. Authorizing rights with this function is most useful for applications that require a one-time authorization. By passing `NULL` to the `authorization` parameter, the Security Server attempts to authorize the requested rights and returns the appropriate result code without actually granting the rights. If you are not going to call any other authorization functions, use this method to determine if a user has authorization without granting any rights.

You can also use this function to preauthorize rights by specifying the `kAuthorizationFlagPreAuthorize` mask. Preauthorization is most useful when a right has a zero timeout. For example, you can preauthorize in the application and if it succeeds, call the helper tool and request authorization. This eliminates calling the helper tool if the user cannot later authorize the specified rights.

If you do not specify the `kAuthorizationFlagPartialRights` mask and the Security Server denies at least one right, then the status of this function on return is `errAuthorizationDenied`.

If you do not specify the `kAuthorizationFlagInteractionAllowed` mask and the Security Server requires user interaction, then the status of this function on return is `errAuthorizationInteractionNotAllowed`.

If you specify the `kAuthorizationFlagInteractionAllowed` mask and the user cancels the authentication process, then the status of this function on return is `errAuthorizationCanceled`.

When your application no longer needs the authorization reference, use the function `AuthorizationFree` (page 17) to free the memory associated with it.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
AuthForAll
BSDLLCTest
MoreIsBetter
QISA

Declared In
`Authorization.h`

AuthorizationCreateFromExternalForm

Internalizes the external representation of an authorization reference.

```
OSStatus AuthorizationCreateFromExternalForm (
   const AuthorizationExternalForm *extForm,
   AuthorizationRef *authorization
);
```

Parameters

extForm

A pointer to the external representation of the authorization reference you retrieve from the calling process.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

authorization

A pointer to an authorization reference. On return, this points to the local copy of the authorization reference. The Security Server allocates the authorization reference for you, so you do not need to call the function `AuthorizationCreate` (page 14).

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
When passing an authorization reference between processes, use this function to internalize the external representation of the authorization reference you created using the function `AuthorizationMakeExternalForm` (page 18).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
MoreIsBetter
QISA

Declared In
`Authorization.h`

AuthorizationExecuteWithPrivileges

Runs an executable tool with root privileges.

```
OSStatus AuthorizationExecuteWithPrivileges (
   AuthorizationRef authorization,
   const char *pathToTool,
   AuthorizationFlags options,
   char *const *arguments,
   FILE **communicationsPipe
);
```

Parameters

authorization

An authorization reference referring to the authorization session.

pathToTool

The full POSIX pathname of the tool to execute.

options

Reserved options. Pass the `kAuthorizationFlagDefaults` constant.

arguments

An argv-style vector of strings to send to the tool.

communicationsPipe

A pointer to a file structure. The Security Server creates the file, opens it for reading and writing, and connects it to the tool's standard input and output. On return, you must close and dispose of this file using `fclose` when your communication is complete. Pass `NULL` if you do not need a communications channel.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
This function enables you to execute the tool you specify in the `pathToTool` parameter as a separate, privileged process. The new process will run with root privileges regardless of the privileges of the invoking process. The new process can retrieve the authorization reference by calling the function `AuthorizationCopyPrivilegedReference` (page 11). The arguments you pass in the `arguments` parameter are relayed to the new process's `argv` parameter. A set of file descriptors is linked to the new process's standard input and output so that your process may communicate with the new process.

To check if the user is authorized to perform this operation, you should preauthorize the `kAuthorizationRightExecute` right. See `AuthorizationItem` (page 23) for a description of what information is included in the authorization item for this right.

Special Considerations

You should use this function only to allow installers to run as root and to allow a `setuid` tool to repair its `setuid` bit if lost. This function works only if the Security Server establishes proper authorization.

This function poses a security concern because it will indiscriminately run any tool or application, severely increasing the security risk. You should avoid the use of this function if possible. One alternative is to split your code into two parts—the application and a setuid tool. The application invokes the setuid tool using standard methods. The setuid tool can then perform the privileged operations. If the tool loses its setuid bit, use the `AuthorizationExecuteWithPrivileges` function to repair it. Factoring your program minimizes the use of this function and reduces the risk of harm. Read *Inside Mac OS X: Performing Privileged Operations With Authorization Services*.

Note that this function respects the setuid bit, if it is set. That is, if the tool you are executing has its setuid bit set and its owner set to foo, the tool will be executed with the user foo's privileges, not root privileges. To ensure that your call to the `AuthorizationExecuteWithPrivileges` function works as intended, make sure the setuid bit of the tool you wish to execute is cleared before calling `AuthorizationExecuteWithPrivileges` to execute the tool.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
MoreIsBetter
QISA

Declared In
`Authorization.h`

AuthorizationFree

Frees the memory associated with an authorization reference.

```
OSStatus AuthorizationFree (
   AuthorizationRef authorization,
   AuthorizationFlags flags
);
```

Parameters

authorization
The authorization reference to free.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

flags

A bit mask. In most cases, pass the constant `kAuthorizationFlagDefaults`. To remove all shared and nonshared authorizations, pass the constant `kAuthorizationFlagDestroyRights`.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
Call this function when your application no longer needs the authorization reference you created using the function `AuthorizationCreate` (page 14).

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
MoreIsBetter
QISA

Declared In
`Authorization.h`

AuthorizationFreeItemSet

Frees the memory associated with an authorization set.

```
OSStatus AuthorizationFreeItemSet (
   AuthorizationItemSet *set
);
```

Parameters

set

A pointer to the authorization set to free.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
When your application no longer needs the authorization item sets created by the Security Server in the AuthorizationCopyRights and AuthorizationCopyInfo functions, you should call this function to free it.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationMakeExternalForm

Creates an external representation of an authorization reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus AuthorizationMakeExternalForm (
   AuthorizationRef authorization,
   AuthorizationExternalForm *extForm
);
```

Parameters

authorization

An authorization reference referring to the authorization session.

extForm

A pointer to an external authorization reference. On return, this points to the external representation of the authorization reference.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
This function creates an external representation of an authorization reference so that you can transmit it between processes. Authorizations are bound by session, process, and time limits, so you cannot store the authorization reference for another process to use. Instead, you must create an external representation of the authorization reference and pass it securely to the other process. Use the function `AuthorizationCreateFromExternalForm` (page 15) to internalize the external representation of the authorization reference.

If it is necessary for your application to perform some privileged operations, it is good programming practice to isolate all of the privileged operations in a separate process, referred to as a *helper tool* (see *Authorization Services Programming Guide* for details). In this case, you must pass your authorization reference to the helper tool so that Authorization Services can tell that the helper tool is operating on behalf of your application. Doing so allows the authorization dialog to show your application's path rather than the path to the helper tool and it allows the system to determine whether the authorization dialog should have keyboard focus.

Availability
Available in Mac OS X v10.0 and later.

Related Sample Code
BSDLLCTest
MoreIsBetter
QISA

Declared In
`Authorization.h`

AuthorizationRightGet

Retrieves a right definition as a dictionary.

```
OSStatus AuthorizationRightGet (
   const char *rightName,
   CFDictionaryRef *rightDefinition
);
```

Parameters

rightName

An ASCII character string representing the rightname. Wildcard right names are valid.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

rightDefinition

A reference to a dictionary. On return, this points to a dictionary of keys that define the right. Passing `NULL` checks if the right is defined. You should release the memory used by the returned dictionary.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
You do not need an authorization reference to use this function because the policy database is world readable.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
AuthForAll

Declared In
`AuthorizationDB.h`

AuthorizationRightRemove

Removes a right from the policy database.

```
OSStatus AuthorizationRightRemove (
   AuthorizationRef authRef,
   const char *rightName
);
```

Parameters

authRef

A valid authorization reference used to authorize modifications.

rightName

An ASCII character string representing the right name. This function does not accept wildcard right names.

Return Value
A result code. See "Authorization Services Result Codes" (page 30).

Discussion
The right you remove must be an explicit right with no wildcards. Wildcard rights are for use by system administrators for site configuration.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`AuthorizationDB.h`

AuthorizationRightSet

Creates or updates a right entry in the policy database.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus AuthorizationRightSet (
   AuthorizationRef authRef,
   const char *rightName,
   CFTypeRef rightDefinition,
   CFStringRef descriptionKey,
   CFBundleRef bundle,
   CFStringRef localeTableName
);
```

Parameters

authRef

A valid authorization reference used to authorize modifications.

rightName

An ASCII character string representing the right name. The policy database does not accept wildcard right names.

rightDefinition

Either a CFDictionary containing keys defining the rules or a CFString representing the name of another right whose rules you wish to duplicate. See `Policy Database Constants` (page 28) for some possible values.

descriptionKey

A CFString reference used as a key for looking up localized descriptions. If no localization is found, this is the description itself. This parameter is optional; pass `NULL` if you do not require it.

bundle

A bundle to get localizations from if not the main bundle. This parameter is optional; pass `NULL` if you do not require it.

localeTableName

A CFString representing a table name from which to get localizations. This parameter is optional; pass `NULL` if you have no localizations or you wish to use the localizations available in Localizable.strings.

Return Value

A result code. See "Authorization Services Result Codes" (page 30).

Discussion

The right you create must be an explicit right with no wildcards. Wildcard rights are for use by system administrators for site configuration.

You can use this function to create a new right or modify an existing right. For example,

```
AuthorizationRightSet(NULL, "com.ifoo.ifax.send",
CFSTR(kAuthorizationRuleIsAdmin), CFSTR("Authorize sending  of a fax"), NULL,
NULL);
```

adds a rule for letting administrators send faxes. This example creates a right named `"com.ifoo.ifax.send"` and sets the rules to require the user to be an administrator by using the `kAuthorizationRuleIsAdmin` constant. This example also sets a comment to let the system administrator know that the right authorizes administrators to send a fax.

To specify additional attributes for the right, you can pass an `CFDictionary` type in the *rightDefinition* parameter as shown in the following example.

```
CFStringRef keys[2] = {CFSTR(kRightRule), CFSTR(kRightComment)};
CFStringRef values[2] = {CFSTR(kAuthorizationRuleIsAdmin), CFSTR("authorizes
sending of 1 fax message")};
CFDictionaryRef aDict;
```


2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
aDict = CFDictionaryCreate(NULL, (void *)keys, (void *)values, 2,
&kCFCopyStringDictionaryKeyCallBacks, &kCFTypeDictionaryValueCallBacks);
AuthorizationRightSet(NULL, "com.ifoo.ifax.send", aDict,  CFSTR("Authorize
sending  of a fax"), NULL, NULL);
CFRelease(aDict);
```

This call creates the same right as before, but adds a specific right comment to the rules definition.

When you specify comments, you should be specific about what you need to authorize. For example, the means of proof required for `kAuthorizationRuleAuthenticateAsAdmin` (a username and password) should not be included here since that rule might be configured differently.

Availability
Available in Mac OS X v10.3 and later.

Related Sample Code
AuthForAll

Declared In
`AuthorizationDB.h`

Data Types

AuthorizationEnvironment

Represents a set of data about the environment, such as user name and other information gathered during evaluation of authorization.

```
typedef AuthorizationItemSet AuthorizationEnvironment;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationExternalForm

The external representation of an authorization reference.

```
struct AuthorizationExternalForm {
    char bytes[kAuthorizationExternalFormLength];
};
```

Fields
`bytes`
An array of characters representing the external form of an authorization reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
Authorization references are bound by session, process, and time limits, so you cannot store the authorization references for another process to use. Use the functions `AuthorizationMakeExternalForm` (page 18) and `AuthorizationCreateFromExternalForm` (page 15) to externalize and internalize the authorization reference. Applications should take care not to disclose the external authorization reference to potential attackers since any process can use this external authorization reference to access the authorization reference.

AuthorizationFlags

Represents a bit mask of authorization options.

```
typedef UInt32 AuthorizationFlags;
```

Discussion
See "Authorization Options" (page 25) for a description of masks that you can use.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationItem

Contains information about an authorization right or the authorization environment.

```
typedef struct {
    AuthorizationString name;
    UInt32 valueLength;
    void *value;
    UInt32 flags;
}AuthorizationItem;
```

Fields

`name`

The required name of the authorization right or environment data. The name of a right is something that you create. You should name rights in a style similar to Java package names. For example, `"com.myOrganization.myProduct.myRight"`. Set this field to `kAuthorizationRightExecute` when requesting a right for use in the function `AuthorizationExecuteWithPrivileges` (page 16).

See the `Security.framework/Headers/AuthorizationTags.h` header file for possible values for environment data.

`valueLength`

An unsigned 32-bit integer that represents the number of bytes in the `value` field. Set the `valueLength` field to `0` if you set the `value` field to `NULL`.

`value`

A pointer to information pertaining to the `name` field. For example, if the `name` field is set to the value represented by the constant `kAuthorizationRightExecute`, then set the `value` field to the full POSIX pathname of the tool you want to execute. In most other cases, set this field to `NULL`.

`flags`

Reserved option bits. Set to `0`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
When using an authorization item to contain a right, set the `name` field to the name of the right—for example, `"com.myOrganization.myProduct.myRight"`, the `valueLength` and `flags` fields to `0`, and the `value` field to `NULL`. For more information on naming rights, read *Authorization Services Programming Guide*

When using an authorization item for the `AuthorizationExecuteWithPrivileges` function, set the `name` field to `kAuthorizationRightExecute`, and the `flags` field to `0`. Set the `value` field to the full POSIX pathname of the tool to execute and the `valueLength` field to the byte length of the value in the `value` field.

When using an authorization item to contain environment data, set the `name` field to the name of the environment data—for example, `kAuthorizationEnvironmentUsername`—and the `flags` field to `0`. Set the `value` field, in this case, to the actual user name and the `valueLength` field to the byte length of the value in the `value` field.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationItemSet

Represents a set of authorization items.

```
typedef struct {
    UInt32 count;
    AuthorizationItem *items;
}AuthorizationItemSet;
```

Fields
`count`
The number of elements in the `items` array.

`items`
A pointer to an array of authorization items. If `count` is greater than `1`, `items` points to the first item in an array of such items. You should set this parameter to `NULL` if there are no items.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationRef

Represents a pointer to an opaque authorization reference structure.

```
typedef const struct AuthorizationOpaqueRef* AuthorizationRef;
```

Discussion
This data type points to a structure the Security Server uses to store information about the authorization session. Use the functions described in "Authorization Services Functions" (page 10) to create, access, and free the authorization reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationRights

Represents a set of authorization rights.

```
typedef AuthorizationItemSet AuthorizationRights;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

AuthorizationString

Represents a zero-terminated string in UTF-8 encoding.

```
typedef const char* AuthorizationString;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`Authorization.h`

Constants

Authorization Options

Define valid authorization options.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
enum {
    kAuthorizationFlagDefaults = 0,
    kAuthorizationFlagInteractionAllowed = (1 << 0),
    kAuthorizationFlagExtendRights = (1 << 1),
    kAuthorizationFlagPartialRights = (1 << 2),
    kAuthorizationFlagDestroyRights = (1 << 3),
    kAuthorizationFlagPreAuthorize = (1 << 4),
    kAuthorizationFlagNoData = (1 << 20)
};
```

Constants

`kAuthorizationFlagDefaults`

If no bits are set, none of the following features are available.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagInteractionAllowed`

If the bit specified by this mask is set, you permit the Security Server to interact with the user when necessary.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagExtendRights`

If the bit specified by this mask is set, the Security Server attempts to grant the rights requested. Once the Security Server denies one right, it ignores the remaining requested rights.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagPartialRights`

If the bit specified by this mask and the `kAuthorizationFlagExtendRights` mask are set, the Security Server grants or denies rights on an individual basis and all rights are checked.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagDestroyRights`

If the bit specified by this mask is set, the Security Server revokes authorization from the process as well as from any other process that is sharing the authorization. If the bit specified by this mask is not set, the Security Server revokes authorization from the process but not from other processes that share the authorization.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagPreAuthorize`

If the bit specified by this mask is set, the Security Server preauthorizes the rights requested.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

`kAuthorizationFlagNoData`

Private bits. Do not use.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
The bits represented by these masks instruct the Security Server how to proceed with the function in which you pass them. Set all unused bits to `0` to allow for future expansion.

Authorization Rights Mask

Defines values the Security Server sets in an authorization item's `flag` field.

```
enum {
    kAuthorizationFlagCanNotPreAuthorize = (1 << 0)
};
```

Constants
`kAuthorizationFlagCanNotPreAuthorize`
Indicates the Security Server could not preauthorize the right.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

Discussion

Empty Environment

Defines an empty environment.

```
#define kAuthorizationEmptyEnvironment NULL
```

Constants
`kAuthorizationEmptyEnvironment`
Indicates an empty environment. You should pass this constant in functions with an environment parameter if you have no environment data to provide.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

External Authorization Reference Length

Defines the byte length of the external authorization reference.

```
enum {
    kAuthorizationExternalFormLength = 32
};
```

Constants
`kAuthorizationExternalFormLength`
Indicates, in number of bytes, the length of the array in the `AuthorizationExternalForm` (page 22) structure.

Available in Mac OS X v10.0 and later.

Declared in `Authorization.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Name Tags

Specify the type of environment data or right.

```
  #define kAuthorizationEnvironmentUsername "username"
#define kAuthorizationEnvironmentPassword "password"
#define kAuthorizationEnvironmentShared "shared"
#define kAuthorizationRightExecute "system.privilege.admin"
#define kAuthorizationEnvironmentPrompt  "prompt"
#define kAuthorizationEnvironmentIcon  "icon"
```

Constants

`kAuthorizationEnvironmentUsername`
Specifies a user name.

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentPassword`
Specifies a password.

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentShared`
Specifies a shared right.

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationRightExecute`
Specifies the name of the right associated with the function `AuthorizationExecuteWithPrivileges` (page 16).

Available in Mac OS X v10.0 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentPrompt`
Specifies the name of the authorization item that should be passed into the environment when specifying invocation-specific additional text. The value should be a localized UTF8 string.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationTags.h`.

`kAuthorizationEnvironmentIcon`
Specifies the name of the authorization item that should be passed into the environment when specifying an alternate icon. The value should be a full path to an image compatible with the `NSImage` class.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationTags.h`.

Discussion

These tags are possible values for the `name` field of an authorization item. This is not an all-inclusive set. You determine the name of the right to request. These environment tags are for future use.

Policy Database Constants

Defines constants for use in settting rights and rules in the policy database.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
#define kAuthorizationRightRule "rule"
#define kAuthorizationRuleIsAdmin "is-admin"
#define kAuthorizationRuleAuthenticateAsAdmin "authenticate-admin"
#define kAuthorizationRuleAuthenticateAsSessionUser "authenticate-session-user"
#define kAuthorizationRuleClassAllow "allow"
#define kAuthorizationRuleClassDeny "deny"
#define kAuthorizationComment "comment"
```

Constants

`kAuthorizationRightRule`

Indicates a rule delegation key. Instead of specifying exact behavior, some rules are shipped with the system and may be used as delegate rules. Use this with any of the delegate rule definition constants.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleIsAdmin`

Indicates a delegate rule definition constant specifying that the user must be an administrator.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleAuthenticateAsAdmin`

Indicates a delegate rule definition constant specifying that the user must authenticate as an administrator.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleAuthenticateAsSessionUser`

Indicates a delegate rule definition constant specifying that the user must authenticate as the session owner (logged-in user).

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleClassAllow`

Indicates a delegate rule definition constant that always allows the specified right.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationRuleClassDeny`

Indicates a deleage rule definition constant that always denies the specified right.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

`kAuthorizationComment`

Indicates comments for a rule. The comments appear in the policy database for the administrator to understand what the rule is for. Rule comments are not the same as localized descriptions which are presented to the user.

Available in Mac OS X v10.3 and later.

Declared in `AuthorizationDB.h`.

Discussion

You can use these constants when creating or modifying a rule in the policy database using the `AuthorizationRightSet` (page 20) function.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Codes

The most common result codes returned by the Security Server are listed in the table below.

Result Code	Value	Description
`errAuthorizationSuccess`	0	The operation completed successfully. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidSet`	-60001	The `set` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidRef`	-60002	The `authorization` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidTag`	-60003	The `tag` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationInvalidPointer`	-60004	The `authorizedRights` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationDenied`	-60005	The Security Server denied authorization for one or more requested rights. This error is also returned if there was no definition found in the policy database, or a definition could not be created. Available in Mac OS X v10.0 and later.
`errAuthorizationCanceled`	-60006	The user canceled the operation. Available in Mac OS X v10.0 and later.
`errAuthorizationInteractionNotAllowed`	-60007	The Security Server denied authorization because no user interaction is allowed. Available in Mac OS X v10.0 and later.
`errAuthorizationInternal`	-60008	An unrecognized internal error occurred. Available in Mac OS X v10.0 and later.
`errAuthorizationExternalizeNotAllowed`	-60009	The Security Server denied externalization of the authorization reference. Available in Mac OS X v10.0 and later.
`errAuthorizationInternalizeNotAllowed`	-60010	The Security Server denied internalization of the authorization reference. Available in Mac OS X v10.0 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
`errAuthorizationInvalidFlags`	-60011	The `flags` parameter is invalid. Available in Mac OS X v10.0 and later.
`errAuthorizationToolExecuteFailure`	-60031	The tool failed to execute. Available in Mac OS X v10.0 and later.
`errAuthorizationToolEnvironmentError`	-60032	The attempt to execute the tool failed to return a success or an error code. Available in Mac OS X v10.0 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Certificate, Key, and Trust Services Reference

Framework:	Security/Security.h
Declared in	SecCertificate.h SecIdentity.h SecIdentitySearch.h SecKey.h SecPolicy.h SecPolicySearch.h SecTrust.h SecTrustedApplication.h SecTrustSettings.h

Overview

Certificate, Key, and Trust Services provides a C API for managing certificates, public and private keys, and trust policies. You can use these services in your application to:

- Determine identity by matching a certificate with a private key
- Create and request certificate objects
- Import certificates, keys, and identities
- Create public-private key pairs
- Represent trust policies

Certificate, Key, and Trust Services can be used in applications running in Aspen.

Functions by Task

Getting Type Identifiers

`SecCertificateGetTypeID` (page 44)
Returns the unique identifier of the opaque type to which a `SecCertificate` object belongs.

`SecIdentityGetTypeID` (page 49)
Returns the unique identifier of the opaque type to which a `SecIdentity` object belongs.

`SecIdentitySearchGetTypeID` (page 51)
Returns the unique identifier of the opaque type to which a `SecIdentitySearch` object belongs.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SecKeyGetTypeID` (page 57)
Returns the unique identifier of the opaque type to which a `SecKey` object belongs.

`SecPolicyGetTypeID` (page 59)
Returns the unique identifier of the opaque type to which a `SecPolicy` object belongs.

`SecPolicySearchGetTypeID` (page 61)
Returns the unique identifier of the opaque type to which a `SecPolicySearch` object belongs.

`SecTrustGetTypeID` (page 70)
Returns the unique identifier of the opaque type to which a `SecTrust` object belongs.

Managing Certificates

`SecCertificateAddToKeychain` (page 37)
Adds a certificate to a keychain.

`SecCertificateCreateFromData` (page 40)
Creates a certificate object based on the specified data, type, and encoding.

`SecCertificateCopyCommonName` (page 38)
Retrieves the common name of the subject of a certificate.

`SecCertificateCopyEmailAddresses` (page 38)
Retrieves the email addresses for the subject of a certificate.

`SecCertificateCopyPreference` (page 39)
Retrieves the preferred certificate for the specified name and key use.

`SecCertificateCopyPublicKey` (page 39)
Retrieves the public key from a certificate.

`SecCertificateGetAlgorithmID` (page 41)
Retrieves the algorithm identifier for a certificate.

`SecCertificateGetCLHandle` (page 41)
Retrieves the certificate library handle from a certificate object.

`SecCertificateGetData` (page 42)
Retrieves the data for a certificate.

`SecCertificateGetIssuer` (page 42)
Unsupported.

`SecCertificateGetItem` (page 43)
Unsupported.

`SecCertificateGetSubject` (page 43)
Unsupported.

`SecCertificateGetType` (page 43)
Retrieves the type of a specified certificate.

`SecCertificateSetPreference` (page 44)
Sets the preferred certificate for a specified name, key use, and date.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Managing Identities

Cryptography and Digital Signatures

Managing Policies

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SecPolicySearchCopyNext` (page 60)
Retrieves a policy object for the next policy matching specified search criteria.

`SecPolicySearchCreate` (page 60)
Creates a search object for finding policies.

`SecPolicySetValue` (page 62)
Sets a policy's value.

Managing Trust

`SecTrustCopyAnchorCertificates` (page 62)
Retrieves the anchor (root) certificates stored by Mac OS X.

`SecTrustCopyCustomAnchorCertificates` (page 63)
Retrieves the custom anchor certificates, if any, used by a given trust.

`SecTrustCopyPolicies` (page 64)
Retrieves the policies used by a given trust management object.

`SecTrustCreateWithCertificates` (page 64)
Creates a trust management object based on certificates and policies.

`SecTrustEvaluate` (page 65)
Evaluates trust for the specified certificate and policies.

`SecTrustGetCssmResult` (page 67)
Retrieves the CSSM trust result.

`SecTrustGetCssmResultCode` (page 68)
Retrieves the CSSM result code from the most recent trust evaluation for a trust management object.

`SecTrustGetResult` (page 69)
Retrieves details on the outcome of a call to the function `SecTrustEvaluate`.

`SecTrustGetTPHandle` (page 70)
Retrieves the trust policy handle.

`SecTrustSetAnchorCertificates` (page 71)
Sets the anchor certificates used when evaluating a trust management object.

`SecTrustSetKeychains` (page 72)
Sets the keychains searched for intermediate certificates when evaluating a trust management object.

`SecTrustSetParameters` (page 73)
Sets the action and action data for a trust management object.

`SecTrustSetPolicies` (page 74)
Set the policies to use in an evaluation.

`SecTrustSetVerifyDate` (page 81)
Sets the date and time against which the certificates in a trust management object are verified.

`SecTrustGetCSSMAnchorCertificates` (page 67) Deprecated in Mac OS X v10.5
Retrieves the CSSM anchor certificates.

`SecTrustGetUserTrust` (page 71) Deprecated in Mac OS X v10.5
Retrieves the user-specified trust setting for a certificate and policy.

`SecTrustSetUserTrust` (page 81) Deprecated in Mac OS X v10.5
Sets the user-specified trust settings of a certificate and policy.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Managing Trust Settings

`SecTrustSettingsCopyCertificates` (page 75)
Obtains an array of all certificates that have trust settings in a specific trust settings domain.

`SecTrustSettingsCopyModificationDate` (page 75)
Obtains the date and time at which a certificate's trust settings were last modified.

`SecTrustSettingsCopyTrustSettings` (page 76)
Obtains the trust settings for a certificate.

`SecTrustSettingsCreateExternalRepresentation` (page 78)
Obtains an external, portable representation of the specified domain's trust settings.

`SecTrustSettingsImportExternalRepresentation` (page 78)
Imports trust settings into a trust domain.

`SecTrustSettingsRemoveTrustSettings` (page 79)
Deletes the trust settings for a certificate.

`SecTrustSettingsSetTrustSettings` (page 80)
Specifies trust settings for a certificate.

Reporting Errors

`SecCopyErrorMessageString` (page 45)
Returns a string describing an error.

Functions

SecCertificateAddToKeychain

Adds a certificate to a keychain.

```
OSStatus SecCertificateAddToKeychain (
   SecCertificateRef certificate,
   SecKeychainRef keychain
);
```

Parameters

certificate
The certificate object for the certificate to add to the keychain.

keychain
The keychain object for the keychain to which you want to add the certificate. Pass `NULL` to add the certificate to the default keychain.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
This function requires a certificate object, which can, for example, be created with the `SecCertificateCreateFromData` (page 40) function or obtained over a network (see *Secure Transport Reference*). If the certificate has already been added to the specified keychain, the function returns

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`errSecDuplicateItem` and does not add another copy to the keychain. The function looks at the certificate data, not at the certificate object, to determine whether the certificate is a duplicate. It considers two certificates to be duplicates if they have the same primary key attributes.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecCertificate.h`

SecCertificateCopyCommonName

Retrieves the common name of the subject of a certificate.

```
OSStatus SecCertificateCopyCommonName(
    SecCertificateRef certificate,
    CFStringRef *commonName
);
```

Parameters

certificate
The certificate object from which to retrieve the common name.

commonName
On return, points to the common name. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecCertificate.h`

SecCertificateCopyEmailAddresses

Retrieves the email addresses for the subject of a certificate.

```
OSStatus SecCertificateCopyEmailAddresses(
    SecCertificateRef certificate,
    CFArrayRef *emailAddresses
);
```

Parameters

certificate
The certificate object from which to retrieve the email addresses.

emailAddresses
On return, an array of zero or more `CFStringRef` elements, each containing one email address found in the certificate subject. Call the `CFRelease` function to release this object when you are finished with it.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
Not every certificate subject includes an email address. If the function does not find any email addresses, it returns a `CFArrayRef` object with zero elements in the array.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecCertificate.h`

SecCertificateCopyPreference

Retrieves the preferred certificate for the specified name and key use.

```
OSStatus SecCertificateCopyPreference(
    CFStringRef name,
    CSSM_KEYUSE keyUsage,
    SecCertificateRef *certificate
);
```

Parameters

name
A string containing an email address (RFC822) or other name for which a preferred certificate is requested.

keyUsage
A key use value, as defined in `Security.framework/cssmtype.h`. Pass `0` to ignore this parameter.

certificate
On return, a reference to the preferred certificate, or `NULL` if none was found. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
This function is typically used to obtain the preferred encryption certificate for an email recipient.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecCertificateSetPreference` (page 44)

Declared In
`SecCertificate.h`

SecCertificateCopyPublicKey

Retrieves the public key from a certificate.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecCertificateCopyPublicKey(
    SecCertificateRef certificate,
    SecKeyRef *key
);
```

Parameters

certificate

The certificate object from which to retrieve the public key.

key

On return, points to the public key for the specified certificate. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecCertificate.h`

SecCertificateCreateFromData

Creates a certificate object based on the specified data, type, and encoding.

```
OSStatus SecCertificateCreateFromData (
   const CSSM_DATA *data,
   CSSM_CERT_TYPE type,
   CSSM_CERT_ENCODING encoding,
   SecCertificateRef *certificate
);
```

Parameters

data

A pointer to the certificate data. The data must be an X509 certificate in binary format.

type

The certificate type as defined in `Security.framework/cssmtype.h`. Permissible values are `CSSM_CERT_X_509v1`, `CSSM_CERT_X_509v2`, and `CSSM_CERT_X_509v3`. If you are unsure of the certificate type, use `CSSM_CERT_X_509v3`.

encoding

The certificate encoding as defined in `Security.framework/cssmtype.h`. Permissible values are `CSSM_CERT_ENCODING_BER` and `CSSM_CERT_ENCODING_DER`. If you are unsure of the encoding, use `CSSM_CERT_ENCODING_BER`.

certificate

On return, points to the newly created certificate object. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The certificate object returned by this function is used as input to several other functions in the API.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecCertificate.h`

SecCertificateGetAlgorithmID

Retrieves the algorithm identifier for a certificate.

```
OSStatus SecCertificateGetAlgorithmID(
    SecCertificateRef certificate,
    const CSSM_X509_ALGORITHM_IDENTIFIER **algid
);
```

Parameters

certificate
The certificate object from which to retrieve the algorithm identifier.

algid
On return, points to a struct that identifies the algorithm for this certificate. This pointer remains valid until the certificate reference is released. Do not attempt to free this pointer.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The `CSSM_X509_ALGORITHM_IDENTIFIER` struct is defined in `Security.framework/x509defs.h` and discussed in *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm). Possible algorithms are enumerated in `Security.framework/oidsalg.h`.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecCertificate.h`

SecCertificateGetCLHandle

Retrieves the certificate library handle from a certificate object.

```
OSStatus SecCertificateGetCLHandle (
   SecCertificateRef certificate,
   CSSM_CL_HANDLE *clHandle
);
```

Parameters

certificate
The certificate object from which to obtain the certificate library handle.

clHandle
On return, points to the certificate library handle of the specified certificate. This handle remains valid until the certificate object is released.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The certificate library handle is the CSSM identifier of the certificate library module that is managing the certificate. The certificate library handle is used as an input to a number of CSSM functions.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecCertificate.h`

SecCertificateGetData

Retrieves the data for a certificate.

```
OSStatus SecCertificateGetData (
   SecCertificateRef certificate,
   CSSM_DATA_PTR data
);
```

Parameters

certificate
A certificate object for the certificate from which to retrieve the data.

data
On return, points to the data for the certificate specified. You must allocate the space for a `CSSM_DATA` structure before calling this function. This data pointer is only guaranteed to remain valid as long as the certificate remains unchanged and valid.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
This function requires a certificate object, which can, for example, be created with the `SecCertificateCreateFromData` (page 40) function, obtained from an identity with the `SecIdentityCopyCertificate` (page 46) function, or obtained over a network (see *Secure Transport Reference*).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecCertificate.h`

SecCertificateGetIssuer

Unsupported.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecCertificateGetIssuer (
   SecCertificateRef certificate,
   CSSM_X509_NAME *issuer
);
```

Availability
Unsupported.

Declared In
SecCertificate.h

SecCertificateGetItem

Unsupported.

```
OSStatus SecCertificateGetItem (
    SecCertificateRef certificate,
    SecKeychainItemRef *item
);
```

Availability
Unsupported.

Declared In
SecCertificate.h

SecCertificateGetSubject

Unsupported.

```
OSStatus SecCertificateGetSubject (
   SecCertificateRef certificate,
   CSSM_X509_NAME *subject
);
```

Availability
Unsupported.

Declared In
SecCertificate.h

SecCertificateGetType

Retrieves the type of a specified certificate.

```
OSStatus SecCertificateGetType (
   SecCertificateRef certificate,
   CSSM_CERT_TYPE *certificateType
);
```

Parameters

certificate

A certificate object for the certificate for which to obtain the type.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

certificateType

On return, points to the type of the specified certificate. Certificate types are defined in `Security.framework/cssmtype.h`. You must allocate the space for a `CSSM_CERT_TYPE` structure before calling this function.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecCertificate.h`

SecCertificateGetTypeID

Returns the unique identifier of the opaque type to which a `SecCertificate` object belongs.

```
CFTypeID SecCertificateGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecCertificateRef` (page 83) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecCertificateRef` (page 83) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecCertificate.h`

SecCertificateSetPreference

Sets the preferred certificate for a specified name, key use, and date.

```
OSStatus SecCertificateSetPreference(
    SecCertificateRef certificate,
    CFStringRef name,
    CSSM_KEYUSE keyUsage,
    CFDateRef date
);
```

Parameters

certificate

The certificate object identifying the preferred certificate.

name

A string containing an email address (RFC822) or other name with which the preferred certificate is to be associated.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

keyUsage

A key use value, as defined in `Security.framework/cssmtype.h`. Pass `0` if you don't want to specify a particular key use.

date

The date after which this preference is no longer valid. If supplied, the preferred certificate is changed only if this date is later than the currently saved setting. Pass `NULL` if this preference should not be restricted by date.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
This function is typically used to set the preferred encryption certificate for an email recipient, either manually (when encrypting email to a recipient) or automatically upon receipt of encrypted email.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecCertificateCopyPreference` (page 39)

Declared In
`SecCertificate.h`

SecCopyErrorMessageString

Returns a string describing an error.

```
CFStringRef SecCopyErrorMessageString(
   OSStatus status,
   void *reserved
);
```

Parameters

status

An error result code of type `OSStatus` or `CSSM_RETURN`, as returned by a security or CSSM function.

reserved

Reserved for future use. Pass `NULL` in this parameter.

Return Value
A reference to an error string, or NULL if no error string is available for the specified result code. You must release this reference when you are finished with it by calling the `CFRelease` function.

Discussion
The error strings returned by this function are taken from the `SecBase.h` header file and are therefore not localizable.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecBase.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecIdentityCopyCertificate

Retrieves a certificate associated with an identity.

```
OSStatus SecIdentityCopyCertificate (
   SecIdentityRef identityRef,
   SecCertificateRef *certificateRef
);
```

Parameters

identityRef

The identity object for the identity whose certificate you wish to retrieve.

certificateRef

On return, points to the certificate object associated with the specified identity.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

An identity is a digital certificate together with its associated private key.

For a certificate in a keychain, you can cast the `SecCertificateRef` data type to a `SecKeychainItemRef` for use with Keychain Services functions.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecIdentity.h`

SecIdentityCopyPreference

Returns the preferred identity for the specified name and key use.

```
OSStatus SecIdentityCopyPreference(
    CFStringRef name,
    CSSM_KEYUSE keyUsage,
    CFArrayRef validIssuers,
    SecIdentityRef *identity
);
```

Parameters

name

A string containing a URI, RFC822 email address, DNS hostname, or other name that uniquely identifies the service requiring an identity.

keyUsage

A key use value, as defined in `Security.framework/cssmtype.h`. Pass `0` if you don't want to specify a particular key use.

validIssuers

An array of `CFDataRef` instances whose contents are the subject names of allowable issuers, as returned by a call to `SSLCopyDistinguishedNames` (`Security.framework/SecureTransport.h`). Pass `NULL` if you don't want to limit the search to specific issuers.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

identity
On return, a reference to the preferred identity, or `NULL` if none was found. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
If a preferred identity has not been set for the specified name, the returned identity reference is `NULL`. You should then typically perform a search for possible identities, using `SecIdentitySearchCreate` (page 50) and `SecIdentitySearchCopyNext` (page 49) , allowing the user to choose from a list if more than one is found.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecIdentitySetPreference` (page 51)

Declared In
`SecIdentity.h`

SecIdentityCopyPrivateKey

Retrieves the private key associated with an identity.

```
OSStatus SecIdentityCopyPrivateKey (
   SecIdentityRef identityRef,
   SecKeyRef *privateKeyRef
);
```

Parameters
identityRef
The identity object for the identity whose private key you wish to retrieve.

privateKeyRef
On return, points to the private key object for the specified identity. The private key must be of class type `kSecAppleKeyItemClass`.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
An identity is a digital certificate together with its associated private key.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecIdentity.h`

SecIdentityCopySystemIdentity

Obtains the system-wide identity associated with a specified domain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecIdentityCopySystemIdentity(
    CFStringRef domain,
    SecIdentityRef *idRef,
    CFStringRef *actualDomain
);
```

Parameters

domain

The domain for which you want to find an identity, typically in reverse DNS notation, such as `com.apple.security`. You may also pass the values defined in "System Identity Domains" (page 90).

idRef

On return, the identity object of the system-wide identity associated with the specified domain. Call the `CFRelease` function to release this object when you are finished with it.

actualDomain

On return, the actual domain name of the returned identity object is returned here. This may be different from the requested domain. Pass `NULL` if you do not want this information.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

If no system-wide identity exists for the specified domain, a domain-specific alternate may be returned instead, typically (but not exclusively) the system-wide default identity (`kSecIdentityDomainDefault`).

Availability

Available in Mac OS X v10.5 and later.

See Also

`SecIdentitySetSystemIdentity` (page 52)

Declared In

`SecIdentity.h`

SecIdentityCreateWithCertificate

Creates a new identity for a certificate and its associated private key.

```
OSStatus SecIdentityCreateWithCertificate(
    CFTypeRef keychainOrArray,
    SecCertificateRef certificateRef,
    SecIdentityRef *identityRef
);
```

Parameters

keychainOrArray

A reference to a keychain or an array of keychains to search for the associated private key. Specify `NULL` to search the user's default keychain search list.

certificateRef

The certificate for which you want to create an identity.

identityRef

On return, an identity object for the certificate and its associated private key. Call the `CFRelease` function to release this object when you are finished with it.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecIdentity.h`

SecIdentityGetTypeID

Returns the unique identifier of the opaque type to which a `SecIdentity` object belongs.

```
CFTypeID SecIdentityGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecIdentityRef` (page 83) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecIdentityRef` (page 83) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecIdentity.h`

SecIdentitySearchCopyNext

Finds the next identity matching specified search criteria

```
OSStatus SecIdentitySearchCopyNext (
   SecIdentitySearchRef searchRef,
   SecIdentityRef *identity
);
```

Parameters

searchRef
An identity search object specifying the search criteria for this search. You create the identity search object by calling the `SecIdentitySearchCreate` (page 50) function.

identity
On return, points to the identity object of the next matching identity (if any). Call the `CFRelease` function to release this object when finished with it.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. When there are no more identities that match the parameters specified to `SecIdentitySearchCreate` (page 50), `errSecItemNotFound` is returned. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecIdentitySearch.h`

SecIdentitySearchCreate

Creates a search object for finding identities.

```
OSStatus SecIdentitySearchCreate (
   CFTypeRef keychainOrArray,
   CSSM_KEYUSE keyUsage,
   SecIdentitySearchRef *searchRef
);
```

Parameters

keychainOrArray
A keychain object for a single keychain to search, an array of keychain objects for a set of keychains to search, or `NULL` to search the user's default keychain search list.

keyUsage
ACSSM key use value as defined in `Security.framework/cssmtype.h`. (Note that, because key recovery is not implemented, the `SIGN_RECOVER` and `VERIFY_RECOVER` constants are not supported.) Use this parameter to filter the search by specifying the key use for the identity. Pass `0` if you want all identities returned by this search. Pass `CSSM_KEYUSE_ANY` to limit the identities returned to those that can be used for every operation.

searchRef
On return, points to the identity search object. Call the `CFRelease` function to release this object when you are done with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
You can OR `CSSM_KEYUSE` values together to set more than one value for key use. Use the returned search object in calls to the `SecIdentitySearchCopyNext` (page 49) function to obtain identities that match the search criteria.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecIdentitySearch.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecIdentitySearchGetTypeID

Returns the unique identifier of the opaque type to which a `SecIdentitySearch` object belongs.

```
CFTypeID SecIdentitySearchGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecIdentitySearchRef` (page 84) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecIdentitySearchRef` (page 84) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecIdentitySearch.h`

SecIdentitySetPreference

Sets the preferred identity for the specified name and key use.

```
OSStatus SecIdentitySetPreference(
    SecIdentityRef identity,
    CFStringRef name,
    CSSM_KEYUSE keyUsage
);
```

Parameters

identity
A reference to the preferred identity.

name
A string containing a URI, RFC822 email address, DNS host name, or other name that uniquely identifies a service requiring this identity.

keyUsage
A key use value, as defined in `Security.framework/cssmtype.h`. Pass `0` if you don't want to specify a particular key use.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecIdentityCopyPreference` (page 46)

Declared In
`SecIdentity.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecIdentitySetSystemIdentity

Assigns the system-wide identity to be associated with a specified domain.

```
OSStatus SecIdentitySetSystemIdentity(
    CFStringRef domain,
    SecIdentityRef idRef
);
```

Parameters

domain

The domain to which the specified identity will be assigned, typically in reverse DNS notation, such as `com.apple.security`. You may also pass the values defined in "System Identity Domains" (page 90).

idRef

The identity to be assigned to the specified domain. Pass `NULL` to delete any currently-assigned identity for the specified domain; in this case, it is not an error if no identity exists for the specified domain.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

The caller must be running as root.

Availability

Available in Mac OS X v10.5 and later.

See Also

`SecIdentityCopySystemIdentity` (page 47)

Declared In

`SecIdentity.h`

SecKeyCreatePair

Creates an asymmetric key pair and stores it in a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeyCreatePair (
   SecKeychainRef keychainRef,
   CSSM_ALGORITHMS algorithm,
   uint32 keySizeInBits,
   CSSM_CC_HANDLE contextHandle,
   CSSM_KEYUSE publicKeyUsage,
   uint32 publicKeyAttr,
   CSSM_KEYUSE privateKeyUsage,
   uint32 privateKeyAttr,
   SecAccessRef initialAccess,
   SecKeyRef *publicKey,
   SecKeyRef *privateKey
);
```

Parameters

keychainRef

The keychain object for the keychain in which to store the private and public key items. Specify `NULL` for the default keychain.

algorithm

The algorithm to use to generate the key pair. Possible values are defined in `Security.framework/cssmtype.h`. Algorithms supported by the AppleCSP module are listed in *Security Release Notes*. This parameter is ignored if the `contextHandle` parameter is not `0`.

keySizeInBits

A key size for the key pair. See *Security Release Notes* for permissible key sizes for each algorithm supported by the AppleCSP module.

contextHandle

A CSSM CSP handle, or `0`. If this argument is not `0`, the `algorithm` and `keySizeInBits` parameters are ignored.

publicKeyUsage

A bit mask indicating all permitted uses for the new public key. The possible values for the `CSSM_KEYUSE` data type are defined in `Security.framework/cssmtype.h`.

publicKeyAttr

A bit mask defining attribute values for the new public key. The bit mask values are equivalent to those defined for `CSSM_KEYATTR_FLAGS` in `Security.framework/cssmtype.h`.

privateKeyUsage

A bit mask indicating all permitted uses for the new private key. The possible values for the `CSSM_KEYUSE` data type are defined in `Security.framework/cssmtype.h`.

privateKeyAttr

A bit mask defining attribute values for the new private key. The bit mask values are defined in `CSSM_KEYATTR_FLAGS` in `Security.framework/cssmtype.h`. Supported values are `CSSM_KETATTR_EXTRACTABLE` (the key can be taken out of the keychain) and `CSSM_KEYATTR_SENSITIVE` (an extractable key can be taken out of the keychain only in wrapped form—that is, encrypted). (Note that you must set *both* of these bits if you want the key to be extractable in wrapped form.) For any other value of this attribute, the key cannot be taken out of the keychain under any circumstances.

initialAccess

An access object that sets the initial access control list for each of the keys returned. See "Creating an Access Object" in *Keychain Services Reference* for functions that create an access object. For default access, specify `NULL`. The default is free access to the tool or application that calls this function, with attempted access to sensitive information by any other application causing a confirmation dialog to be displayed.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

publicKey

On return, points to the keychain item object of the new public key. Use this object as input to the `SecKeyGetCSSMKey` (page 57) function to obtain the `CSSM_KEY` structure containing the key. Call the `CFRelease` function to release this object when you are finished with it.

privateKey

On return, points to the keychain item object of the new private key. Use this object as input to the `SecKeyGetCSSMKey` (page 57) function to obtain the `CSSM_KEY` structure containing the key. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
This function uses default values for any attributes required by specific key-generation algorithms. Algorithms supported by the AppleCSP module are listed in *Security Release Notes*. For details about algorithms and default values for key-generation parameters, download the CDSA security framework from the ADC website at http://developer.apple.com/darwin/projects/security/ and read the file `Supported_CSP_Algorithms.doc` in the Documentation folder.

If you need extra parameters to generate a key—as required by some algortihms—call `SecKeychainGetCSPHandle` (page 127) to obtain a CSSM CSP handle and then call `CSSM_CSP_CreateKeyGenContext` to create a context. With this context, use `CSSM_UpdateContextAttributes` to add additional parameters. Finally, call `CSSM_DeleteContext` to dispose of the context after calling this function.

Availability
Available in Mac OS X v10.2 and later.

See Also
`SecKeyGenerate` (page 54)

Declared In
`SecKey.h`

SecKeyGenerate

Creates a symmetric key and optionally stores it in a keychain.

```
OSStatus SecKeyGenerate(
    SecKeychainRef keychainRef,
    CSSM_ALGORITHMS algorithm,
    uint32 keySizeInBits,
    CSSM_CC_HANDLE contextHandle,
    CSSM_KEYUSE keyUsage,
    uint32 keyAttr,
    SecAccessRef initialAccess,
    SecKeyRef* keyRef
);
```

Parameters

keychainRef

The keychain in which to store the generated key. Specify `NULL` to generate a transient key.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

algorithm
The algorithm to use in generating the symmetric key. Possible values are defined in `cssmtype.h`. Algorithms supported by the AppleCSP module are listed in *Security Release Notes*. This parameter is ignored if the `contextHandle` parameter is not `0`.

keySizeInBits
A key size for the key pair. This parameter is ignored if the `contextHandle` parameter is not `0`.

contextHandle
A CSSM CSP handle, or `0`. If this argument is not `0`, the `algorithm` and `keySizeInBits` parameters are ignored.

keyUsage
A bit mask indicating all permitted uses for the new key. The possible values for the `CSSM_KEYUSE` data type are defined in `cssmtype.h`.

keyAttr
A bit mask defining attribute values for the new key. The bit mask values are defined in `CSSM_KEYATTR_FLAGS` in `cssmtype.h`.

initialAccess
An access object that sets the initial access control list for the key returned. See "Creating an Access Object" in *Keychain Services Reference* for functions that create an access object. This parameter is ignored if you specify `NULL` for the `keychainRef` parameter.

keyRef
On return, points to the keychain item object of the new public key. Use this object as input to the `SecKeyGetCSSMKey` (page 57) function to obtain the `CSSM_KEY` structure containing the key. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
Key-generation algorithms supported by the AppleCSP module are listed in *Security Release Notes*. For details about algorithms and default values for key-generation parameters, download the CDSA security framework from the ADC website at http://developer.apple.com/darwin/projects/security/ and read the file `Supported_CSP_Algorithms.doc` in the Documentation folder.

If you need extra parameters to generate a key—as required by some algortihms—call `SecKeychainGetCSPHandle` (page 127) to obtain a CSSM CSP handle and then call `CSSM_CSP_CreateKeyGenContext` to create a context. With this context, use `CSSM_UpdateContextAttributes` to add additional parameters. Finally, call `CSSM_DeleteContext` to dispose of the context after calling this function.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecKeyCreatePair` (page 52)

Declared In
`SecKey.h`

SecKeyGetCredentials

Returns an access credential for a key.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeyGetCredentials(
    SecKeyRef keyRef,
    CSSM_ACL_AUTHORIZATION_TAG operation,
    SecCredentialType credentialType,
    const CSSM_ACCESS_CREDENTIALS **outCredentials
);
```

Parameters

keyRef

The key for which you want an access credential.

operation

The type of operation to be performed with this key. Possible values are listed under "Authorization tag types" in `Security.framework/cssmtype.h`.

credentialType

The type of credential requested. See "Key Credential Type Constants" (page 91) for possible values.

outCredentials

On return, points to an access credential for the specified key. This pointer remains valid until the key reference is released. Do not attempt to modify or free this data.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
An access credential is required as an input to a number of CSSM functions.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecKey.h`

SecKeyGetCSPHandle

Returns the CSSM CSP handle for a key.

```
OSStatus SecKeyGetCSPHandle(
    SecKeyRef keyRef,
    CSSM_CSP_HANDLE *cspHandle
);
```

Parameters

keyRef

The key for which you want a CSSM CSP handle.

cspHandle

On return, points to the CSSM CSP handle for the specified key. This pointer remains valid until the key reference is released. Do not attempt to modify or free this data.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
A CSSM CSP handle is required as an input to a number of CSSM functions.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.5 and later.

Declared In
SecKey.h

SecKeyGetCSSMKey

Retrieves a pointer to the CSSM_KEY structure containing the key stored in a keychain item.

```
OSStatus SecKeyGetCSSMKey (
   SecKeyRef key,
   const CSSM_KEY **cssmKey
);
```

Parameters

key
A keychain key item object.

cssmKey
A pointer to a CSSM_KEY structure for the specified key. You should not modify or free this data, because it is owned by the system.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The CSSM_KEY structure is used to represent keys in CSSM and is used as an input value to several CSSM functions. The CSSM_KEY structure is valid until the keychain item object is released.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKey.h

SecKeyGetTypeID

Returns the unique identifier of the opaque type to which a SecKey object belongs.

```
CFTypeID SecKeyGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a SecKeyRef (page 84) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a SecKeyRef (page 84) object. You can compare this value to the CFTypeID identifier obtained by calling the CFGetTypeID function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecKey.h

SecPolicyGetOID

Retrieves a policy's object identifier.

```
OSStatus SecPolicyGetOID (
   SecPolicyRef policyRef,
   CSSM_OID *oid
);
```

Parameters

policyRef

The policy object for which to obtain the object identifier. You can obtain a policy object with the SecPolicySearchCopyNext (page 60) function.

oid

On return, points to the policy's object identifier. This identifier is owned by the policy object and remains valid until that object is destroyed; do not release it separately.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The policy's object identifier, in the form of a CSSM_OID structure, is used in the CSSM API together with the policy's value. Use the SecPolicyGetValue (page 59) function to obtain the value that corresponds to this object identifier.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecPolicy.h

SecPolicyGetTPHandle

Retrieves the trust policy handle for a policy object.

```
OSStatus SecPolicyGetTPHandle (
   SecPolicyRef policyRef,
   CSSM_TP_HANDLE *tpHandle
);
```

Parameters

policyRef

The policy object from which to obtain the trust policy handle.

tpHandle

On return, points to the policy object's trust policy handle. The handle remains valid until the policy object is released.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
The trust policy handle is the CSSM identifier of the trust policy module that is managing the certificate. The trust policy handle is uses as an input to a number of CSSM functions.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecPolicy.h`

SecPolicyGetTypeID

Returns the unique identifier of the opaque type to which a `SecPolicy` object belongs.

```
CFTypeID SecPolicyGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecPolicyRef` (page 84) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecPolicyRef` (page 84) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecPolicy.h`

SecPolicyGetValue

Retrieves a policy's value.

```
OSStatus SecPolicyGetValue (
   SecPolicyRef policyRef,
   CSSM_DATA *value
);
```

Parameters

policyRef

The policy object for which to retrieve the value.

value

On return, points to the policy's value. This value is owned by the policy object and remains valid until that object is destroyed; do not release it separately.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
A policy's value is defined and interpreted by the policy. If you are using CSSM, you can specify object-identifier–policy-value pairs as input to the `CSSM_TP_POLICYINFO` function. Use the `SecPolicyGetOID` (page 58) function to obtain the object identifier (OID) for a policy.

Depending on how the policy uses the value, the value can be specific to a transaction. Because some other process might be using this policy object, it is best not to assign a new value to the policy using the same policy object. Instead, obtain a new policy object before assigning a new value to the policy.

Availability
Available in Mac OS X v10.2 and later.

See Also
`SecPolicySetValue` (page 62)

Declared In
`SecPolicy.h`

SecPolicySearchCopyNext

Retrieves a policy object for the next policy matching specified search criteria.

```
OSStatus SecPolicySearchCopyNext (
   SecPolicySearchRef searchRef,
   SecPolicyRef *policyRef
);
```

Parameters

searchRef
A policy search object specifying the search criteria for this search. You create the policy search object by calling the `SecPolicySearchCreate` (page 60) function.

policyRef
On return, points to the policy object for the next policy (if any) matching the specified search criteria. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. When there are no more policies that match the parameters specified to `SecPolicySearchCreate` (page 60), `errSecPolicyNotFound` is returned. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecPolicySearch.h`

SecPolicySearchCreate

Creates a search object for finding policies.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecPolicySearchCreate (
   CSSM_CERT_TYPE certType,
   const CSSM_OID *policyOID,
   const CSSM_DATA *value,
   SecPolicySearchRef *searchRef
);
```

Parameters

certType

The type of certificates a policy uses, as defined in `Security.framework/cssmtype.h`. Permissible values are `CSSM_CERT_X_509v1`, `CSSM_CERT_X_509v2`, and `CSSM_CERT_X_509v3`. If you are unsure of the certificate type, use `CSSM_CERT_X_509v3`.

policyOID

A pointer to a BER-encoded policy object identifier that uniquely specifies the policy. See "AppleX509TP Trust Policies" for a list of policies and object identifiers provided by the AppleX509TP module.

value

A pointer to an optional, policy-defined value. The contents of this value depend on the policy object identifier specified. (Note that this parameter refers to the value stored in MDS and is not related to the `value` parameter of the `SecPolicyGetValue` (page 59) function.) Currently the function does not use this parameter; pass `NULL` for this pointer.

searchRef

On return, points to the newly created policy search object. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
You use the search object created by this function in subsequent calls to the `SecPolicySearchCopyNext` (page 60) function to obtain trust policy objects. Policies are stored in the Module Directory Services (MDS) database. MDS is described in detail in "Part 8: Module Directory Service (MDS)" of *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecPolicySearch.h`

SecPolicySearchGetTypeID

Returns the unique identifier of the opaque type to which a `SecPolicySearch` object belongs.

```
CFTypeID SecPolicySearchGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecPolicySearchRef` (page 85) object.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecPolicySearchRef` (page 85) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecPolicySearch.h`

SecPolicySetValue

Sets a policy's value.

```
OSStatus SecPolicySetValue(
    SecPolicyRef policyRef,
    const CSSM_DATA *value
);
```

Parameters

policyRef
The policy object whose value you wish to set.

value
The value to be set into the policy object, replacing any previous value.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
A policy's value is defined and interpreted by the policy. If you are using CSSM, you can specify object-identifier–policy-value pairs as input to the `CSSM_TP_POLICYINFO` function. Use the `SecPolicyGetOID` (page 58) function to obtain the object identifier (OID) for a policy.

Depending on how the policy uses the value, the value can be specific to a transaction. Because some other process might be using this policy object, it is best not to assign a new value to the policy using the same policy object. Instead, obtain a new policy object before assigning a new value to the policy.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecPolicyGetValue` (page 59)

Declared In
`SecPolicy.h`

SecTrustCopyAnchorCertificates

Retrieves the anchor (root) certificates stored by Mac OS X.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecTrustCopyAnchorCertificates (
   CFArrayRef *anchors
);
```

Parameters

anchors

On return, points to an array of certificate objects for trusted anchor (root) certificates, which is the default set of anchors for the caller. Call the `CFRelease` function to release the `CFArrayRef` object when you are finished with it.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

This function retrieves the certificates in the system's store of anchor certificates (see `SecTrustSetAnchorCertificates` (page 71)). You can use the `SecCertificateRef` objects retrieved by this function as input to other functions of this API, such as `SecTrustCreateWithCertificates` (page 64). If you want references to the anchor certificates in a form appropriate for calls to the CSSM API, use the `SecTrustGetCSSMAnchorCertificates` (page 67) function instead.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecTrust.h`

SecTrustCopyCustomAnchorCertificates

Retrieves the custom anchor certificates, if any, used by a given trust.

```
OSStatus SecTrustCopyCustomAnchorCertificates(
    SecTrustRef trust,
    CFArrayRef *anchors
);
```

Parameters

trust

The trust management object from which you wish to retrieve the custom anchor certificates.

anchors

On return, a reference to an array of `SecCertificateRef` objects representing the set of anchor certificates that are considered valid (trusted) anchors by the `SecTrustEvaluate` (page 65) function when verifying a certificate using the trust management object in the `trust` parameter. Returns `NULL` if no custom anchors have been specified. Call the `CFRelease` function to release this object when you are finished with it.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

You can use the `SecTrustSetAnchorCertificates` (page 71) function to set custom anchor certificates.

Availability

Available in Mac OS X v10.5 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

See Also
SecTrustSetAnchorCertificates (page 71)

Declared In
SecTrust.h

SecTrustCopyPolicies

Retrieves the policies used by a given trust management object.

```
OSStatus SecTrustCopyPolicies(
    SecTrustRef trust,
    CFArrayRef *policies
);
```

Parameters

trust

The trust management object whose policies you wish to retrieve.

policies

On return, an array of SecPolicyRef (page 84) objects for the policies used by this trust management object. Call the CFRelease function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability
Available in Mac OS X v10.5 and later.

See Also
SecTrustSetPolicies (page 74)

Declared In
SecTrust.h

SecTrustCreateWithCertificates

Creates a trust management object based on certificates and policies.

```
OSStatus SecTrustCreateWithCertificates (
   CFArrayRef certificates,
   CFTypeRef policies,
   SecTrustRef *trustRef
);
```

Parameters

certificates

The certificate to be verified, plus any other certificates you think might be useful for verifying the certificate. The certificate to be verified must be the first in the array. If you want to specify only one certificate, you can pass a SecCertificateRef object; otherwise, pass an array of SecCertificateRef objects.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

policies

References to one or more policies to be evaluated. You can pass a single `SecPolicyRef` object, or an array of one or more `SecPolicyRef` objects. Use the `SecPolicySearchCopyNext` (page 60) function to obtain policy objects. If you pass in multiple policies, all policies must verify for the certificate chain to be considered valid.

trustRef

On return, points to the newly created trust management object. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The trust management object includes a reference to the certificate to be verified, plus pointers to the policies to be evaluated for those certificates. You can optionally include references to other certificates, including anchor certificates, that you think might be in the certificate chain needed to verify the first (leaf) certificate. Any input certificates that turn out to be irrelevant are harmlessly ignored. Call the `SecTrustEvaluate` (page 65) function to evaluate the trust for the returned trust management object.

If not all the certificates needed to verify the leaf certificate are included in the `certificates` parameter, `SecTrustEvaluate` searches for certificates in the keychain search list (see `SecTrustSetKeychains` (page 72)) and in the system's store of anchor certificates (see `SecTrustSetAnchorCertificates` (page 71)). However, you should gain a significant performance benefit by passing in the entire certificate chain, in order, in the `certificates` parameter.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrust.h`

SecTrustEvaluate

Evaluates trust for the specified certificate and policies.

```
OSStatus SecTrustEvaluate (
   SecTrustRef trust,
   SecTrustResultType *result
);
```

Parameters

trust

The trust management object to evaluate. A trust management object includes the certificate to be verified plus the policy or policies to be used in evaluating trust. It can optionally also include other certificates to be used in verifying the first certificate. Use the `SecTrustCreateWithCertificates` (page 64) function to create a trust management object.

result

On return, points to a result type reflecting the result of this evaluation. See "Trust Result Type Constants" (page 88) for descriptions of possible values.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
This function evaluates a certificate's validity to establish trust for a particular use—for example, in creating a digital signature or to establish a Secure Sockets Layer connection. Before you call this function, you can optionally call any of the `SecTrustSet...` functions (such as `SecTrustSetParameters` (page 73) or `SecTrustSetVerifyDate` (page 81)) to set values for parameters and options.

The `SecTrustEvaluate` function validates a certificate by verifying its signature plus the signatures of the certificates in its certificate chain, up to the anchor certificate, according to the policy or policies included in the trust management object. For each policy, the function evaluates trust according to the user-specified trust setting (see `SecTrustSetUserTrust` (page 81) and `SecTrustGetUserTrust` (page 71)). For an example of user-specified trust settings, use the Keychain Access utility and look at any certificate.

For each policy, `SecTrustEvaluate` starts with the leaf certificate and checks each certificate in the chain, in turn, for a valid user-specified trust setting. It uses the first such value it finds for the trust evaluation. For example, if the user-specified trust for the leaf certificate is not set, the first intermediate certificate is set to "Always Trust," and one of the other intermediate certificates is set to "Never Trust," `SecTrustEvaluate` trusts the certificate. Thus, you can use a user-specified trust setting for a certificate closer to the leaf to override a setting closer to the anchor.

If there is no user-specified trust setting for the entire certificate chain, the `SecTrustEvaluate` function returns `kSecTrustResultUnspecified` as the result type. In that case, you should call the `SFCertificateTrustPanel` class in the *Security Interface Framework Reference* to let the user specify a trust setting for the certificate. Alternately, you can use a default value. If you use a default value, you should provide a preference setting so that the user can change the default.

If `SecTrustEvaluate` returns `kSecTrustResultRecoverableTrustFailure` as the result type, you can call the `SecTrustGetResult` (page 69) function for details of the problem. Then, as appropriate, you can call one or more of the `SecTrustSet...` functions to correct or bypass the problem, or you can inform the user of the problem and call the `SFCertificateTrustPanel` class to let the user change the trust setting for the certificate. When you think you have corrected the problem, call `SecTrustEvaluate` again. Each time you call `SecTrustEvaluate`, it discards the results of any previous evaluation and replaces them with the new results. If `SecTrustEvaluate` returns `kSecTrustResultFatalTrustFailure`, on the other hand, changing parameter values and calling `SecTrustEvaluate` again is unlikely to be successful.

If not all the certificates needed to verify the leaf certificate are included in the trust management object, then `SecTrustEvaluate` searches for certificates in the keychain search list (see `SecTrustSetKeychains` (page 72)) and in the system's store of anchor certificates (see `SecTrustSetAnchorCertificates` (page 71)).

By default, `SecTrustEvaluate` uses the current date and time when verifying a certificate. However, you can call the `SecTrustSetVerifyDate` (page 81) function before calling `SecTrustEvaluate` to set an other date and time to use when verifying the certificate.

Before you call `SecTrustEvaluate`, you can optionally use the `SecTrustSetParameters` (page 73) function to set one or more actions to modify the evaluation or to pass data required by an action.

The results of the trust evaluation are stored in the trust management object. Call the `SecTrustGetResult` (page 69) function to get more information about the results of the trust evaluation, or the `SecTrustGetCssmResult` (page 67) function to get information about the evaluation in a form that can be passed to CSSM functions.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecTrust.h

SecTrustGetCSSMAnchorCertificates

Retrieves the CSSM anchor certificates. (Deprecated in Mac OS X v10.5.)

```
OSStatus SecTrustGetCSSMAnchorCertificates (
   const CSSM_DATA **cssmAnchors,
   uint32 *cssmAnchorCount
);
```

Parameters

cssmAnchors

On return, points to an array of anchor certificates. This array is allocated by the system; you should not deallocate it. This data is not guaranteed to remain valid indefinitely; you should retrieve the data immediately and either pass it to other functions or copy it for future use.

cssmAnchorCount

On return, points to the number of CSSM_DATA structures returned in the cssmAnchors parameter.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
This function returns the certificates in the system's store of anchor certificates (see SecTrustSetAnchorCertificates (page 71). You can use the CSSM_DATA structures returned by this function as input to functions in the CSSM API. If you want references to the anchor certificates in a form appropriate for calls to the Certificate, Key, and Trust API, use the SecTrustCopyAnchorCertificates (page 62) function instead.

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Declared In
SecTrust.h

SecTrustGetCssmResult

Retrieves the CSSM trust result.

```
OSStatus SecTrustGetCssmResult (
   SecTrustRef trust,
   CSSM_TP_VERIFY_CONTEXT_RESULT_PTR *result
);
```

Parameters

trust

A trust management object that has previously been sent to the SecTrustEvaluate (page 65) function for evaluation.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

result

On return, points to the CSSM trust result pointer. You should not modify or free this data, as it is owned by the system.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
After calling the `SecTrustEvaluate` (page 65) function, you can call the `SecTrustGetResult` (page 69) function or the `SecTrustGetCssmResult` function to get information about the certificates in the certificate chain and everything that might be wrong with each certificate. Whereas the `SecTrustGetResult` (page 69) function returns the information in a form that you can interpret without extensive knowledge of CSSM, the `SecTrustGetCssmResult` function returns information in a form that can be passed directly to CSSM functions. See *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm for more information about the `CSSM_TP_VERIFY_CONTEXT_RESULT` structure pointed to by the `result` parameter.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrust.h`

SecTrustGetCssmResultCode

Retrieves the CSSM result code from the most recent trust evaluation for a trust management object.

```
OSStatus SecTrustGetCssmResultCode(
    SecTrustRef trust,
    OSStatus *resultCode
);
```

Parameters

trust

The trust management object for which you wish to retrieve a result code.

resultCode

On return, the CSSM result code produced by the most recent call to the `SecTrustEvaluate` (page 65) function for the trust management object specified in the `trust` parameter. The value of this parameter is undefined if `SecTrustEvaluate` has not been called.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95). Returns `errSecTrustNotAvailable` if the `SecTrustEvaluate` function has not been called for the specified trust.

Discussion
Whereas the `SecTrustEvaluate` function returns one of the Security Framework result codes (see "Certificate, Key, and Trust Services Result Codes" (page 95)), the `SecTrustGetCssmResultCode` function returns the CSSM result code as enumerated in `Security.framework/cssmerr.h`. Other functions that might be of interest are the `SecTrustGetResult` (page 69) function, which returns detailed results for each certificate in the certificate chain, and the `SecTrustGetCssmResult` (page 67) function, which returns the results in a format that can be passed directly to CSSM functions.

Availability
Available in Mac OS X v10.5 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

See Also
SecTrustEvaluate (page 65)

Declared In
SecTrust.h

SecTrustGetResult

Retrieves details on the outcome of a call to the function SecTrustEvaluate.

```
OSStatus SecTrustGetResult (
   SecTrustRef trustRef,
   SecTrustResultType *result,
   CFArrayRef *certChain,
   CSSM_TP_APPLE_EVIDENCE_INFO **statusChain
);
```

Parameters

trustRef

A trust management object that has previously been sent to the SecTrustEvaluate (page 65) function for evaluation.

result

A pointer to the result type returned in the result parameter by the SecTrustEvaluate function.

certChain

On return, points to an array of certificates that constitute the certificate chain used to verify the input certificate. Call the CFRelease function to release this object when you are finished with it.

statusChain

On return, points to an array of CSSM_TP_APPLE_EVIDENCE_INFO structures, one for each certificate in the certificate chain. The first item in the array corresponds to the leaf certificate, and the last item corresponds to the anchor (assuming that verification of the chain did not fail before reaching the anchor certificate). Each structure describes the status of one certificate in the chain. This structure is defined in cssmapple.h. Do not attempt to free this pointer; it remains valid until the trust management object is released or until the next call to the function SecTrustEvaluate that uses this trust management object.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
After calling the SecTrustEvaluate (page 65) function, you can call the SecTrustGetResult function or the SecTrustGetCssmResult (page 67) function to get detailed information about the results of the evaluation. Whereas the SecTrustGetResult function returns the information in a form that you can interpret without extensive knowledge of CSSM, the SecTrustGetCssmResult (page 67) function returns information in a form that can be passed directly to CSSM functions.

You can call the SFCertificateTrustPanel class in the *Security Interface Framework Reference* to display these results to the user.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecTrust.h

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecTrustGetTPHandle

Retrieves the trust policy handle.

```
OSStatus SecTrustGetTPHandle (
   SecTrustRef trust,
   CSSM_TP_HANDLE *handle
);
```

Parameters

trust

The trust management object from which to obtain the trust policy handle. A trust management object includes one or more certificates plus the policy or policies to be used in evaluating trust. Use the `SecTrustCreateWithCertificates` (page 64) function to create a trust management object.

handle

On return, points to a CSSM trust policy handle. This handle remains valid until the trust management object is released or until the next call to the function `SecTrustEvaluate` (page 65) that uses this trust management object.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

The trust policy handle is the CSSM identifier of the trust policy module that is managing the certificate. The trust policy handle is used as an input to a number of CSSM functions.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecTrust.h`

SecTrustGetTypeID

Returns the unique identifier of the opaque type to which a `SecTrust` object belongs.

```
CFTypeID SecTrustGetTypeID (
   void
);
```

Return Value

A value that identifies the opaque type of a `SecTrustRef` (page 85) object.

Discussion

This function returns a value that uniquely identifies the opaque type of a `SecTrustRef` (page 85) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecTrust.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecTrustGetUserTrust

Retrieves the user-specified trust setting for a certificate and policy. (Deprecated in Mac OS X v10.5.)

```
OSStatus SecTrustGetUserTrust (
   SecCertificateRef certificate,
   SecPolicyRef policy,
   SecTrustUserSetting *trustSetting
);
```

Parameters

certificate

The certificate object from which to obtain the user-specified trust setting.

policy

The policy object for the policy for which to obtain the user-specified trust setting. Use the `SecPolicySearchCopyNext` (page 60) function to obtain a policy object.

trustSetting

On return, points to the user-specified trust setting for the specified certificate and policy.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

Each certificate has one user-specified trust setting per policy. For each policy, the user can specify that the certificate is always to be trusted, is never to be trusted, or can be trusted only after permission is requested from—and granted by—the user. It is also possible for there to be no user-specified trust setting for a policy. See `SecTrustEvaluate` (page 65) for a discussion of the use of user-specified trust settings in a trust evaluation.

The `SecTrustGetUserTrust` function returns the effective user trust setting for the certificate and policy specified. You can obtain a certificate from a keychain and typecast the keychain item object (data type `SecKeychainItemRef`) to a certificate object (`SecCertificateRef`).

See "Trust Result Type Constants" (page 88) for values and descriptions of the user-specified trust settings. The user can set these values in the Keychain Access utility. If you provide your own UI for these settings, you can use the `SecTrustSetUserTrust` (page 81) function to set them.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.5.

Declared In

`SecTrust.h`

SecTrustSetAnchorCertificates

Sets the anchor certificates used when evaluating a trust management object.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecTrustSetAnchorCertificates (
   SecTrustRef trust,
   CFArrayRef anchorCertificates
);
```

Parameters

trust

The trust management object containing the certificate you want to evaluate. A trust management object includes the certificate to be verified plus the policy or policies to be used in evaluating trust. It can optionally also include other certificates to be used in verifying the first certificate. Use the `SecTrustCreateWithCertificates` (page 64) function to create a trust management object.

anchorCertificates

A reference to an array of `SecCertificateRef` objects representing the set of anchor certificates that are to be considered valid (trusted) anchors by the `SecTrustEvaluate` (page 65) function when verifying a certificate. Pass `NULL` to restore the default set of anchor certificates.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

The `SecTrustEvaluate` (page 65) function looks for an anchor certificate in the array of certificates specified by the `SecTrustSetAnchorCertificates` function, or uses a default set provided by the system. In Mac OS X v10.3, for example, the default set of anchors is in the keychain file /System/Library/Keychains/X509Anchors. If you want to create a set of anchor certificates by modifying the default set, call the `SecTrustCopyAnchorCertificates` (page 62) function to obtain the current set of anchor certificates, modify that set as you wish, and create a new array of certificates. Then call `SecTrustSetAnchorCertificates` with the modified array.

The list of custom anchor certificates is stored in the trust management object and can be retrieved with the `SecTrustCopyCustomAnchorCertificates` (page 63) function.

Use the `SecTrustSetKeychains` (page 72) function to set the keychains searched for intermediate certificates in the certificate chain.

Important: Calling this function without also calling `SecTrustSetAnchorCertificatesOnly` disables the trusting of any anchors other than the ones specified by this function call.

Availability

Available in Mac OS X v10.2 and later.

See Also

`SecTrustCopyCustomAnchorCertificates` (page 63)

Declared In

`SecTrust.h`

SecTrustSetKeychains

Sets the keychains searched for intermediate certificates when evaluating a trust management object.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecTrustSetKeychains (
   SecTrustRef trust,
   CFTypeRef keychainOrArray
);
```

Parameters

trust

The trust management object containing the certificate you want to evaluate. A trust management object includes the certificate to be verified plus the policy or policies to be used in evaluating trust. It can optionally also include other certificates to be used in verifying the first certificate. Use the `SecTrustCreateWithCertificates` (page 64) function to create a trust management object.

keychainOrArray

A keychain object for a single keychain to search, an array of keychain objects for a set of keychains to search, or `NULL` to search the user's default keychain search list. To prevent the `SecTrustEvaluate` (page 65) function from searching any keychains at all, pass a `CFArrayRef` array with no elements.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

By default, `SecTrustEvaluate` (page 65) uses the user's keychain search list to look for intermediate certificates in the certificate chain. Use the `SecTrustSetKeychains` function to change the set of keychains to be searched. If you want to modify the default set of keychains, first call the `SecKeychainCopySearchList` function (see *Keychain Services Reference*) to obtain the current keychain search list, modify that set as you wish, and create a new search list. Then you can call `SecTrustSetKeychains` with the modified list.

Use the `SecTrustSetAnchorCertificates` (page 71) function to set the array of anchor certificates searched.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecTrust.h`

SecTrustSetParameters

Sets the action and action data for a trust management object.

```
OSStatus SecTrustSetParameters (
   SecTrustRef trustRef,
   CSSM_TP_ACTION action,
   CFDataRef actionData
);
```

Parameters

trustRef

The trust management object to which you want to add an action or set action data. A trust management object includes one or more certificates plus the policy or policies to be used in evaluating trust. Use the `SecTrustCreateWithCertificates` (page 64) function to create a trust management object.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

action

A CSSM trust action. Pass `CSSM_TP_ACTION_DEFAULT` for the default action. Other actions available, if any, are described in the documentation for the trust policy module. For the AppleX509TP module, see the *Security Release Notes*.

actionData

A reference to action data. "Action Data Flags" (page 89) lists possible values for this parameter for the AppleX509TP trust policy module's default action. For other actions (if any), the possible values for the action data are specified in the *Security Release Notes*.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
Before you call `SecTrustEvaluate` (page 65), you can optionally use this function to set one or more action flags or to set action data. Actions, where available, affect the trust evaluation for all policies being evaluated. For example, if you set the action data for the default action to `CSSM_TP_ACTION_ALLOW_EXPIRED`, then the `SecTrustEvaluate` function ignores the certificate's expiration date and time.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrust.h`

SecTrustSetPolicies

Set the policies to use in an evaluation.

```
OSStatus SecTrustSetPolicies(
    SecTrustRef trust,
    CFTypeRef policies
);
```

Parameters

trust

The trust management object whose policy list you wish to set.

policies

An array of one or more `SecPolicyRef` (page 84) objects for the policies to be used by this trust management object. A single policy object of type `SecPolicyRef` may also be passed, representing an array of one policy.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
The policies you set with this function replace any already in the trust management object.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecTrustCopyPolicies` (page 64)

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
`SecTrust.h`

SecTrustSettingsCopyCertificates

Obtains an array of all certificates that have trust settings in a specific trust settings domain.

```
OSStatus SecTrustSettingsCopyCertificates(
    SecTrustSettingsDomain domain,
    CFArrayRef *certArray
);
```

Parameters

domain

The trust settings domain for which you want a list of certificates. For possible values, see "Trust Settings Domain Constants" (page 91).

certArray

On return, an array of `SecCertificateRef` objects representing the certificates that have trust settings in the specified domain. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95). Returns `errSecNoTrustSettings` if no trust settings exist for the specified domain.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecTrustSettings.h`

SecTrustSettingsCopyModificationDate

Obtains the date and time at which a certificate's trust settings were last modified.

```
OSStatus SecTrustSettingsCopyModificationDate(
    SecCertificateRef certRef,
    SecTrustSettingsDomain domain,
    CFDateRef *modificationDate
);
```

Parameters

certRef

The certificate for which you wish to obtain the modification time. Pass the value `kSecTrustSettingsDefaultRootCertSetting` to obtain the modification time for the default root certificate trust settings for the domain.

domain

The trust settings domain of the trust settings for which you wish to obtain the modification time. For possible values, see "Trust Settings Domain Constants" (page 91).

modificationDate

On return, the date and time at which the certificate's trust settings were last modified. Call the `CFRelease` function to release this object when you are finished with it.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95). Returns `errSecItemNotFound` if no trust settings exist for the specified certificate and domain.

Availability
Available in Mac OS X v10.5 and later.

Declared In
`SecTrustSettings.h`

SecTrustSettingsCopyTrustSettings

Obtains the trust settings for a certificate.

```
OSStatus SecTrustSettingsCopyTrustSettings(
    SecCertificateRef certRef,
    SecTrustSettingsDomain domain,
    CFArrayRef *trustSettings
);
```

Parameters

certRef
The certificate for which you want the trust settings. Pass the value `kSecTrustSettingsDefaultRootCertSetting` to obtain the default root certificate trust settings for the domain.

domain
The trust settings domain of the trust settings that you wish to obtain. For possible values, see "Trust Settings Domain Constants" (page 91).

trustSettings
On return, an array of `CFDictionary` objects specifying the trust settings for the certificate. For the contents of the dictionaries, see the discussion below. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95). Returns `errSecItemNotFound` if no trust settings exist for the specified certificate and domain.

Discussion
Each certificate's trust settings are expressed as a `CFArray` that includes any number (including zero) of dictionaries of type `CFDictionary`, each of which comprises one set of usage constraints. Each usage constraints dictionary contains zero or one of each of the following key-value pairs:

Key	Value
`kSecTrustSettingsPolicy`	A policy object (`SecPolicyRef`) specifying the certificate verification policy; for example: SSL, SMIME. Use the `SecPolicySearchCopyNext` (page 60) function to obtain a policy object.
`kSecTrustSettingsApplication`	A trusted application reference (`SecTrustedApplicationRef`) for the application checking the certificate's trust settings. Use the `SecTrustedApplicationCreateFromPath` (page 154) function to get this reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Key	Value
kSecTrustSettingsPolicyString	A CFString containing policy-specific data. For the SMIME policy, this string contains an email address. For the SSL policy, it contains a host name.
kSecTrustSettingsKeyUsage	A CFNumber containing an SInt32 value specifying the operations for which the encryption key in this certificate can be used. For possible values, see "Trust Settings Key Use Constants" (page 92).
kSecTrustSettingsResult	A CFNumber containing an SInt32 value indicating the effective trust setting for this usage constraints dictionary. A given usage constraints dictionary is included in the evaluation of trust for a certificate only if the specified policy, application, and key use match the use for which the certificate is being evaluated. If this is the case, then the value of the kSecTrustSettingsResult key is ORed with the results from other dictionaries to determine the overall trust setting for the certificate. If this key is not present, a default value of kSecTrustSettingsResultTrustRoot is assumed. Because only a root certificate can have this value, a usage constraints dictionary for a non-root certificate that is missing this key is not valid. Possible values for this key are listed in "Trust Settings Result Constants" (page 94).
kSecTrustSettingsAllowedError	A CFNumber containing an SInt32 value indicating a CSSM_RETURN result code which, if encountered during certificate verification, is ignored for that certificate. These "allowed error" values are applied to the evaluation only if the usage constraints dictionary meets the criteria described with the kSecTrustSettingsResult key. A usage constraint dictionary with no constraints but with an allowed error value causes that error to always be allowed when the certificate is being evaluated.

The overall trust settings for a certificate are the sum of all the usage constraints dictionaries that match the use for which that certificate is being evaluated. Trust settings for a given use apply if *any* of the dictionaries in the certificate's trust settings array satisfies the specified use. Thus, when a certificate has multiple usage constraints dictionaries in its trust settings array, the overall trust settings for the certificate are:

((usage constraint dictionary 0 component 0) AND (usage constraint dictionary 0 component 1) AND (...)) OR ((usage constraint dictionary 1 component 0) AND (usage constraint dictionary 1 component 1) AND (...)) OR (...) ...

If the value of the kSecTrustSettingsResult component is *not* kSecTrustSettingsResultUnspecified for a usage constraints dictionary that has no constraints, the default value kSecTrustSettingsResultTrustRoot is assumed. To specify a value for the kSecTrustSettingsAllowedError component without explicitly trusting or distrusting the associated certificate, specify a value of kSecTrustSettingsResultUnspecified for the kSecTrustSettingsResult component. An empty trust settings array (that is, the trustSettings parameter returns a valid but empty CFArray) means "always trust this certificate" with an overall trust setting for the certificate of kSecTrustSettingsResultTrustRoot. Note that an empty trust settings array is not the same as no trust settings (the trustSettings parameter returns NULL), which means "this certificate must be verified to a known trusted certificate". Note the distinction between the results kSecTrustSettingsResultTrustRoot and kSecTrustSettingsResultTrustAsRoot: The former can only be applied to root (self-signed) certificates; the latter can only be applied to non-root certificates. Therefore, an empty trust settings array

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

for a non-root certificate is invalid, because the default value of `kSecTrustSettingsResultTrustRoot` is not valid for a non-root certificate. When making changes to the per-user trust settings, the user is prompted with an alert panel asking for authentication (user name and password or other credentials normally used for login). Therefore, it is not possible to modify per-user trust settings when not running in a GUI environment (that is, when the user is not logged in via the login window). When making changes to the system-wide trust settings, the user is prompted with an alert panel asking for an administrator's name and password unless the calling process is running as root, in which case no futher authentication is needed.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecTrustSettingsSetTrustSettings` (page 80)

Declared In
`SecTrustSettings.h`

SecTrustSettingsCreateExternalRepresentation

Obtains an external, portable representation of the specified domain's trust settings.

```
OSStatus SecTrustSettingsCreateExternalRepresentation(
    SecTrustSettingsDomain domain,
    CFDataRef *trustSettings
);
```

Parameters

domain

The trust settings domain for which you want an external representation of trust settings. For possible values, see "Trust Settings Domain Constants" (page 91).

trustSettings

An external representation of the domain's trust settings. Call the `CFRelease` function to release this object when you are finished with it.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95). Returns `errSecNoTrustSettings` if no trust settings exist for the specified domain.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecTrustSettingsImportExternalRepresentation` (page 78)

Declared In
`SecTrustSettings.h`

SecTrustSettingsImportExternalRepresentation

Imports trust settings into a trust domain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecTrustSettingsImportExternalRepresentation(
    SecTrustSettingsDomain domain,
    CFDataRef trustSettings
);
```

Parameters

domain

The trust settings domain into which you want to import trust settings. For possible values, see "Trust Settings Domain Constants" (page 91).

trustSettings

An external representation of the trust settings (created by the `SecTrustSettingsCreateExternalRepresentation` (page 78) function) that you want to import.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Availability

Available in Mac OS X v10.5 and later.

See Also

`SecTrustSettingsCreateExternalRepresentation` (page 78)

Declared In

`SecTrustSettings.h`

SecTrustSettingsRemoveTrustSettings

Deletes the trust settings for a certificate.

```
OSStatus SecTrustSettingsRemoveTrustSettings(
    SecCertificateRef certRef,
    SecTrustSettingsDomain domain);
```

Parameters

certRef

The certificate whose trust settings you wish to remove. Pass the value `kSecTrustSettingsDefaultRootCertSetting` to remove the default root certificate trust settings for the domain.

domain

The trust settings domain for which you wish to remove the trust settings. For possible values, see "Trust Settings Domain Constants" (page 91).

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95). Returns `errSecItemNotFound` if no trust settings exist for the certificate.

Discussion

If a certificate has no trust settings, the certificate must be verified to a known, trusted certificate.

Availability

Available in Mac OS X v10.5 and later.

See Also

`SecTrustSettingsSetTrustSettings` (page 80)

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecTrustSettings.h

SecTrustSettingsSetTrustSettings

Specifies trust settings for a certificate.

```
OSStatus SecTrustSettingsSetTrustSettings(
    SecCertificateRef certRef,
    SecTrustSettingsDomain domain,
    CFTypeRef trustSettingsDictOrArray);
```

Parameters

certRef

The certificate for which you want to specify the trust settings. Pass the value `kSecTrustSettingsDefaultRootCertSetting` to set the default root certificate trust settings for the domain.

domain

The trust settings domain of the trust settings that you wish to specify. For possible values, see "Trust Settings Domain Constants" (page 91).

trustSettings

On return, an array of `CFDictionary` objects specifying the trust settings for the certificate. For the contents of the dictionaries, see the discussion below. Call the `CFRelease` function to release this object when you are finished with it.

trustSettingsDictOrArray

The trust settings you wish to specify for this certificate, in the form of a `CFDictionary` object, a `CFArray` of `CFDictionary` objects, or `NULL`. The contents of `CFDictionary` objects used to specify trust settings are detailed in the `SecTrustSettingsCopyTrustSettings` (page 76) function description. Pass `NULL` if you want to specify an empty trust settings array.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion
If you pass `NULL` for the `trustSettingsDictOrArray` parameter, then the trust settings for this certificate are stored as an empty trust settings array, indicating "always trust this root certificate regardless of use." This setting is valid only for a self-signed (root) certificate. To instead remove all trust settings for the certificate (interpreted as "this certificate must be verified to a known trusted certificate"), use the `SecTrustSettingsRemoveTrustSettings` (page 79) function.

If the specified certificate already has trust settings in the specified domain, this function replaces them.

Availability
Available in Mac OS X v10.5 and later.

See Also
`SecTrustSettingsCopyTrustSettings` (page 76)
`SecTrustSettingsRemoveTrustSettings` (page 79)

Declared In
SecTrustSettings.h

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecTrustSetUserTrust

Sets the user-specified trust settings of a certificate and policy. (Deprecated in Mac OS X v10.5.)

```
OSStatus SecTrustSetUserTrust (
   SecCertificateRef certificate,
   SecPolicyRef policy,
   SecTrustUserSetting trustSetting
);
```

Parameters

certificate

The certificate object for which to set the user-specified trust settings. Use the `SecCertificateCreateFromData` (page 40) function to obtain a certificate object.

policy

The policy object for the policy for which to set the user-specified trust settings. Use the `SecPolicySearchCopyNext` (page 60) function to obtain a policy object.

trustSetting

The user-specified trust setting to be set. See "Trust Result Type Constants" (page 88) for possible values.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

Each certificate has one user-specified trust setting per policy. These trust settings are used by the `SecTrustEvaluate` (page 65) function when evaluating trust. See "Trust Result Type Constants" (page 88) for values and descriptions of the user-specified trust settings. The user can set these values in the Keychain Access utility. Under certain circumstances, it might be appropriate for an administrative application to change a user trust setting. In that case, you can use the `SecTrustSetUserTrust` function to do so. You can obtain a certificate from a keychain and typecast the keychain item object (data type `SecKeychainItemRef`) to a certificate object (`SecCertificateRef`).

When you call the `SecTrustSetUserTrust` function, the user might be prompted to confirm the new setting before it is changed.

You can use the `SecTrustGetUserTrust` (page 71) function to get the current user-specified trust settings for a certificate.

Availability

Available in Mac OS X v10.2 and later.

Deprecated in Mac OS X v10.5.

Declared In

`SecTrust.h`

SecTrustSetVerifyDate

Sets the date and time against which the certificates in a trust management object are verified.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecTrustSetVerifyDate (
   SecTrustRef trust,
   CFDateRef verifyDate
);
```

Parameters

trust

The trust management object whose verification date you want to set. A trust management object includes one or more certificates plus the policy or policies to be used in evaluating trust. Use the `SecTrustCreateWithCertificates` (page 64) function to create a trust management object.

verifyDate

The date and time to use when verifying the certificate.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 95).

Discussion

By default, the `SecTrustEvaluate` (page 65) function uses the current date and time when verifying a certificate. However, you can use `SecTrustSetVerifyDate` to set another date and time to use when verifying a certificate. For example, you can determine whether the certificate was valid when the document was signed rather than whether it's valid at the present time.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecTrust.h`

Data Types

CSSM_TP_APPLE_EVIDENCE_INFO

Contains information about a certificate evaluation.

```
typedef struct {
    CSSM_TP_APPLE_CERT_STATUS   StatusBits;
    uint32                      NumStatusCodes
    CSSM_RETURN                 *StatusCodes;
    uint32                      Index;
    CSSM_DL_DB_HANDLE           DlDbHandle
    CSSM_DB_UNIQUE_RECORD_PTR   UniqueRecord;
} CSSM_TP_APPLE_EVIDENCE_INFO;
```

Fields

`StatusBits`

Indicates whether the certificate is valid and where it was found; see "Certificate Status Constants" (page 87).

`NumStatusCodes`

The number of `CSSM_RETURN` structures returned in the `StatusCodes` field.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`StatusCodes`
: An array of `CSSM_RETURN` values indicating what problems were found with the certificate. Apple-specific values are in `cssmapple.h`. Standard CSSM values are defined in `cssmerr.h` and are discussed in "Error Codes and Error Values" in the "Trust Policy Services API" chapter of *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm

`Index`
: An index into the standard set of certificates or anchor certificates if the certificate came from one of those sets.

`DlDbHandle`
: A CSSM object that identifies a particular database. This field is used if the certificate did not come from the standard set of certificates or anchor certificates. This value is useful only as input to functions in the CSSM API.

`UniqueRecord`
: A CSSM object that identifies a particular record in a database. This field is used if the certificate did not come from the standard set of certificates or anchor certificates. This value is useful only as in input to functions in the CSSM API.

Discussion
An array of these structures is returned by the `SecTrustGetResult` (page 69) function; each one describes a certificate in the certificate chain.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`cssmapple.h`

SecCertificateRef

Abstract Core Foundation-type object representing an X.509 certificate.

```
typedef struct __SecCertificate *SecCertificateRef;
```

Discussion
A `SecCertificateRef` object for a certificate that is stored in a keychain can be safely cast to a `SecKeychainItemRef` for manipulation as a keychain item. On the other hand, if the `SecCertificateRef` is not stored in a keychain, casting the object to a `SecKeychainItemRef` and passing it to Keychain Services functions returns errors.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

SecIdentityRef

Abstract Core Foundation-type object representing an identity.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef struct __SecIdentity *SecIdentityRef;
```

Discussion
A `SecIdentityRef` object contains a `SecKeyRef` object and an associated `SecCertificateRef` object.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

SecIdentitySearchRef

Contains information about an identity search.

```
typedef struct OpaqueSecIdentitySearchRef *SecIdentitySearchRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecIdentitySearch.h`

SecKeyRef

Abstract Core Foundation-type object representing an asymmetric key.

```
typedef struct __SecKey *SecKeyRef;
```

Discussion
A `SecKeyRef` object for a key that is stored in a keychain can be safely cast to a `SecKeychainItemRef` for manipulation as a keychain item. On the other hand, if the `SecKeyRef` is not stored in a keychain, casting the object to a `SecKeychainItemRef` and passing it to Keychain Services functions returns errors.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

SecPolicyRef

Contains information about a policy.

```
typedef struct OpaqueSecPolicyRef *SecPolicyRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecPolicySearchRef

Contains information about a policy search.

```
typedef struct OpaquePolicySearchRef *SecPolicySearchRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecPolicySearch.h`

SecPublicKeyHash

Represents a 20-byte public key hash.

```
typedef UInt8 SecPublicKeyHash[20];
```

Discussion
The `SecPublicKeyHash` type represents a hash of a public key. You can use the constant `kSecPublicKeyHashItemAttr` as input to functions in the Keychain Services API to set or retrieve a certificate attribute value of this type. See *Keychain Services Reference* for information about getting and setting attribute values.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecTrustRef

Contains information about trust management.

```
typedef struct __SecTrust *SecTrustRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrust.h`

SecTrustUserSetting

Represents user-specified trust settings.

```
typedef SecTrustResultType SecTrustUserSetting;
```

Discussion
See "Trust Result Type Constants" (page 88) for possible values.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
`SecTrust.h`

Constants

Certificate Item Attribute Constants

Indicates certificate item attributes.

```
enum
{
    kSecSubjectItemAttr              = 'subj',
    kSecIssuerItemAttr               = 'issu',
    kSecSerialNumberItemAttr         = 'snbr',
    kSecPublicKeyHashItemAttr        = 'hpky',
    kSecSubjectKeyIdentifierItemAttr = 'skid',
    kSecCertTypeItemAttr             = 'ctyp',
    kSecCertEncodingItemAttr         = 'cenc'
};
```

Constants

`kSecSubjectItemAttr`
DER-encoded subject distinguished name.
Available in Mac OS X v10.2 and later.
Declared in `SecCertificate.h`.

`kSecIssuerItemAttr`
DER-encoded issuer distinguished name.
Available in Mac OS X v10.2 and later.
Declared in `SecCertificate.h`.

`kSecSerialNumberItemAttr`
DER-encoded certificate serial number.
Available in Mac OS X v10.2 and later.
Declared in `SecCertificate.h`.

`kSecPublicKeyHashItemAttr`
Public key hash.
Available in Mac OS X v10.2 and later.
Declared in `SecCertificate.h`.

`kSecSubjectKeyIdentifierItemAttr`
Subject key identifier.
Available in Mac OS X v10.2 and later.
Declared in `SecCertificate.h`.

`kSecCertTypeItemAttr`
Certificate type.
Available in Mac OS X v10.2 and later.
Declared in `SecCertificate.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecCertEncodingItemAttr`
Certificate encoding.

Available in Mac OS X v10.2 and later.

Declared in `SecCertificate.h`.

Certificate Status Constants

Specifies the status of a certificate.

```
typedef uint32 CSSM_TP_APPLE_CERT_STATUS;
enum
{
    CSSM_CERT_STATUS_EXPIRED            = 0x00000001,
    CSSM_CERT_STATUS_NOT_VALID_YET      = 0x00000002,
    CSSM_CERT_STATUS_IS_IN_INPUT_CERTS  = 0x00000004,
    CSSM_CERT_STATUS_IS_IN_ANCHORS      = 0x00000008,
    CSSM_CERT_STATUS_IS_ROOT            = 0x00000010,
    CSSM_CERT_STATUS_IS_FROM_NET        = 0x00000020
};
```

Constants

`CSSM_CERT_STATUS_EXPIRED`
The certificate has expired.

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

`CSSM_CERT_STATUS_NOT_VALID_YET`
The certificate is not yet valid. In addition to the expiration, or "Not Valid After," date and time, each certificate has a "Not Valid Before" date and time.

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

`CSSM_CERT_STATUS_IS_IN_INPUT_CERTS`
This is one of the certificates included in the array of certificates passed to the `SecTrustCreateWithCertificates` (page 64) function.

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

`CSSM_CERT_STATUS_IS_IN_ANCHORS`
This certificate was found in the system's store of anchor certificates (see `SecTrustSetAnchorCertificates` (page 71)).

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

`CSSM_CERT_STATUS_IS_ROOT`
The certificate is a root certificate. If this bit is set but the `CSSM_CERT_STATUS_IS_IN_ANCHORS` bit is not, then this is an untrusted anchor.

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`CSSM_CERT_STATUS_IS_FROM_NET`
The certificate was obtained through some mechanism other than the certificates stored by the operating system and those passed into the `SecTrustCreateWithCertificates` (page 64) function. For example, the certificate might have been fetched over a network.

Available in Mac OS X v10.3 and later.

Declared in `cssmapple.h`.

Discussion
If none of these bits are set, the certificate came from a standard certificate search; see the description of the `SecTrustSetKeychains` (page 72) function.

Trust Result Type Constants

Specifies the trust result type.

```
typedef enum {
    kSecTrustResultInvalid,
    kSecTrustResultProceed,
    kSecTrustResultConfirm,
    kSecTrustResultDeny,
    kSecTrustResultUnspecified,
    kSecTrustResultRecoverableTrustFailure,
    kSecTrustResultFatalTrustFailure,
    kSecTrustResultOtherError
} SecTrustResultType;
```

Constants
`kSecTrustResultInvalid`
Invalid setting or result. Usually, this result indicates that the `SecTrustEvaluate` (page 65) function did not complete successfully.

Available in Mac OS X v10.2 and later.

Declared in `SecTrust.h`.

`kSecTrustResultProceed`
The user indicated that you may trust the certificate for the purposes designated in the specified policies. This value may be returned by the `SecTrustEvaluate` (page 65) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed "Always Trust."

Available in Mac OS X v10.2 and later.

Declared in `SecTrust.h`.

`kSecTrustResultConfirm`
Confirmation from the user is required before proceeding. This value may be returned by the `SecTrustEvaluate` (page 65) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed "Ask Permission."

Available in Mac OS X v10.2 and later.

Declared in `SecTrust.h`.

`kSecTrustResultDeny`
The user specified that the certificate should not be trusted. This value may be returned by the `SecTrustEvaluate` (page 65) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed "Never Trust."

Available in Mac OS X v10.2 and later.

Declared in `SecTrust.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecTrustResultUnspecified`
: The user did not specify a trust setting. This value may be returned by the `SecTrustEvaluate` (page 65) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed "Use System Policy." This is the default user setting.

  Available in Mac OS X v10.2 and later.

  Declared in `SecTrust.h`.

`kSecTrustResultRecoverableTrustFailure`
: Trust denied; retry after changing settings. For example, if trust is denied because the certificate has expired, you can ask the user whether to trust the certificate anyway. If the user answers yes, then use the `SecTrustSetUserTrust` (page 81) function to set the user trust setting to `kSecTrustResultProceed` and call `SecTrustEvaluate` (page 65) again. This value may be returned by the `SecTrustEvaluate` (page 65) function but not stored as part of the user trust settings.

  Available in Mac OS X v10.2 and later.

  Declared in `SecTrust.h`.

`kSecTrustResultFatalTrustFailure`
: Trust denied; no simple fix is available. For example, if a certificate cannot be verified because it is corrupted, trust cannot be established without replacing the certificate. This value may be returned by the `SecTrustEvaluate` (page 65) function but not stored as part of the user trust settings.

  Available in Mac OS X v10.2 and later.

  Declared in `SecTrust.h`.

`kSecTrustResultOtherError`
: A failure other than that of trust evaluation; for example, an internal failure of the `SecTrustEvaluate` (page 65) function. This value may be returned by the `SecTrustEvaluate` (page 65) function but not stored as part of the user trust settings.

  Available in Mac OS X v10.2 and later.

  Declared in `SecTrust.h`.

Discussion
These constants may be returned by the `SecTrustEvaluate` (page 65) function or stored as one of the user trust settings (see `SecTrustSetUserTrust` (page 81)), as noted. When evaluating user trust, both `SecTrustGetUserTrust` (page 71) and `SecTrustEvaluate` start with the leaf certificate and work through the chain down to the anchor. The `SecTrustGetUserTrust` function returns the user trust setting of the first certificate for which the setting is other than `kSecTrustResultUnspecified`. Similarly, the function uses the user trust setting of the first certificate for which the setting is other than `kSecTrustResultUnspecified`, regardless of the user trust settings of other certificates in the chain.

Action Data Flags

Specifies options for the AppleX509TP trust policy module's default action.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef uint32 CSSM_APPLE_TP_ACTION_FLAGS;
enum {
    CSSM_TP_ACTION_ALLOW_EXPIRED        = 0x00000001,
    CSSM_TP_ACTION_LEAF_IS_CA           = 0x00000002,
    CSSM_TP_ACTION_FETCH_CERT_FROM_NET  = 0x00000004,
    CSSM_TP_ACTION_ALLOW_EXPIRED_ROOT   = 0x00000008
};
```

Constants

`CSSM_TP_ACTION_ALLOW_EXPIRED`

Ignore the expiration date and time for all certificates.

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

`CSSM_TP_ACTION_LEAF_IS_CA`

First certificate is that of a certification authority (CA). By formal definition, a valid certificate chain must begin with a certificate that is not a CA. Set this bit if you want to validate a partial chain, starting with a CA and working toward the anchor, or if you want to evaluate a single self-signed certificate as a one-certificate "chain" for testing purposes.

Available in Mac OS X v10.3 and later.

Declared in `cssmapple.h`.

`CSSM_TP_ACTION_FETCH_CERT_FROM_NET`

Enable fetching intermediate certificates over the network using http or LDAP.

Available in Mac OS X v10.3 and later.

Declared in `cssmapple.h`.

`CSSM_TP_ACTION_ALLOW_EXPIRED_ROOT`

Ignore the expiration date and time for root certificates only.

Available in Mac OS X v10.2 and later.

Declared in `cssmapple.h`.

Discussion

See `SecTrustSetParameters` (page 73) for more information about actions.

System Identity Domains

Domains for which you can set or obtain a system-wide identity.

```
const CFStringRef kSecIdentityDomainDefault;
const CFStringRef kSecIdentityDomainKerberosKDC;
```

Constants

`kSecIdentityDomainDefault`

The system-wide default identity.

Available in Mac OS X v10.5 and later.

Declared in `SecIdentity.h`.

`kSecIdentityDomainKerberosKDC`

Kerberos Key Distribution Center (KDC) identity.

Available in Mac OS X v10.5 and later.

Declared in `SecIdentity.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
These constants can be used with the `SecIdentitySetSystemIdentity` (page 52) and `SecIdentityCopySystemIdentity` (page 47) functions.

Key Credential Type Constants

The credential type to be returned by `SecKeyGetCredentials` (page 55).

```
typedef uint32 SecCredentialType;

enum
{
    kSecCredentialTypeDefault = 0,
    kSecCredentialTypeWithUI,
    kSecCredentialTypeNoUI
};
```

Constants

`kSecCredentialTypeDefault`
The default setting for determining whether to present UI is used.

The default setting can be changed with a call to `SecKeychainSetUserInteractionAllowed` (page 152).

Available in Mac OS X v10.5 and later.

Declared in `SecKey.h`.

`kSecCredentialTypeWithUI`
Keychain operations on keys that have this credential are allowed to present UI if required.

Available in Mac OS X v10.5 and later.

Declared in `SecKey.h`.

`kSecCredentialTypeNoUI`
Keychain operations on keys that have this credential are not allowed to present UI, and will fail if UI is required.

Available in Mac OS X v10.5 and later.

Declared in `SecKey.h`.

Discussion
See the section "Servers and the Keychain" in the Keychain Services Tasks chapter of *Keychain Services Programming Guide* for information on the use of UI with keychain tasks.

Trust Settings Domain Constants

The trust settings domains used by the trust settings API.

```
enum {     kSecTrustSettingsDomainUser = 0,     kSecTrustSettingsDomainAdmin,
 kSecTrustSettingsDomainSystem }; typedef uint32 SecTrustSettingsDomain;
```

Constants

`kSecTrustSettingsDomainUser`
Per-user trust settings.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecTrustSettingsDomainAdmin`
Locally administered, system-wide trust settings.

Administrator privileges are required to make changes to this domain.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

`kSecTrustSettingsDomainSystem`
System trust settings.

These trust settings are immutable and comprise the set of trusted root certificates supplied in Mac OS X. These settings are read-only, even by root.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

Trust Settings Key Use Constants

Allowed uses for the encryption key in a certificate.

```
enum {
    kSecTrustSettingsKeyUseSignature        = 0x00000001,
    kSecTrustSettingsKeyUseEnDecryptData    = 0x00000002,
    kSecTrustSettingsKeyUseEnDecryptKey     = 0x00000004,
    kSecTrustSettingsKeyUseSignCert         = 0x00000008,
    kSecTrustSettingsKeyUseSignRevocation   = 0x00000010,
    kSecTrustSettingsKeyUseKeyExchange      = 0x00000020,
    kSecTrustSettingsKeyUseAny              = 0xffffffff
};
typedef uint32 SecTrustSettingsKeyUsage;
```

Constants

`kSecTrustSettingsKeyUseSignature`
The key can be used to sign data or verify a signature.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

`kSecTrustSettingsKeyUseEnDecryptData`
The key can be used to encrypt or decrypt data.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

`kSecTrustSettingsKeyUseEnDecryptKey`
The key can be used to encrypt or decrypt (wrap or unwrap) a key.

Private keys must be wrapped before they can be exported from a keychain.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

`kSecTrustSettingsKeyUseSignCert`
The key can be used to sign a certificate or verify a signature.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecTrustSettingsKeyUseSignRevocation`
: The key can be used to sign an OCSP (online certificate status protocol) message or CRL (certificate verification list), or to verify a signature.

  OCSP messages and CRLs are used to revoke certificates.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsKeyUseKeyExchange`
: The key is a private key that has been shared using a key exchange protocol, such as Diffie-Hellman key exchange.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsKeyUseAny`
: The key can be used for any purpose.

  This is the default key-use setting if no other key use is specified.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

Trust Settings Usage Constraints Dictionary Keys

The keys in one usage constraints dictionary.

```
#define kSecTrustSettingsPolicy          CFSTR("kSecTrustSettingsPolicy")
#define kSecTrustSettingsApplication     CFSTR("kSecTrustSettingsApplication")
#define kSecTrustSettingsPolicyString    CFSTR("kSecTrustSettingsPolicyString")
#define kSecTrustSettingsKeyUsage        CFSTR("kSecTrustSettingsKeyUsage")
#define kSecTrustSettingsAllowedError    CFSTR("kSecTrustSettingsAllowedError")
#define kSecTrustSettingsResult          CFSTR("kSecTrustSettingsResult")
```

Constants

`kSecTrustSettingsPolicy`
: A policy object (`SecPolicyRef`) specifying the certificate verification policy.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsApplication`
: A trusted application reference (`SecTrustedApplicationRef`) for the application checking the certificate's trust settings.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsPolicyString`
: A`CFString` containing policy-specific data.

  For the SMIME policy, this string contains an email address. For the SSL policy, it contains a host name.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecTrustSettingsKeyUsage`
: A `CFNumber` containing an `SInt32` value specifying the operations for which the encryption key in this certificate can be used.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsAllowedError`
: A `CFNumber` containing an `SInt32` value indicating a `CSSM_RETURN` result code which, if encountered during certificate verification, is ignored for that certificate.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsResult`
: A `CFNumber` containing an `SInt32` value indicating the effective trust setting for this usage constraints dictionary.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

Trust Settings Result Constants

Effective trust settings for usage constraints dictionaries used by the `SecTrustSettingsCopyTrustSettings` (page 76) and `SecTrustSettingsSetTrustSettings` (page 80) functions.

```
enum {
    kSecTrustSettingsResultInvalid = 0,
    kSecTrustSettingsResultTrustRoot,
    kSecTrustSettingsResultTrustAsRoot,
    kSecTrustSettingsResultDeny,
    kSecTrustSettingsResultUnspecified
};
typedef uint32 SecTrustSettingsResult;
```

Constants

`kSecTrustSettingsResultInvalid`
: Never valid in a trust settings array or in an API call.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsResultTrustRoot`
: This root certificate is explicitly trusted.

  If the certificate is not a root (self-signed) certificate, the usage constraints dictionary is invalid.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

`kSecTrustSettingsResultTrustAsRoot`
: This non-root certificate is explicitly trusted as if it were a trusted root.

  Available in Mac OS X v10.5 and later.

  Declared in `SecTrustSettings.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecTrustSettingsResultDeny`
This certificate is explicitly distrusted.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

`kSecTrustSettingsResultUnspecified`
This certificate is neither trusted nor distrusted. This value can be used to specify an "allowed error" without assigning trust to a specific certificate.

This value can be used to specify an allowed error without assigning trust to the certificate.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

Default Root Certificate Trust Settings

A value indicating the default root certificate trust settings when used for a `SecCertificateRef` object in a trust settings API function.

```
#define kSecTrustSettingsDefaultRootCertSetting        ((SecCertificateRef)-1)
```

Constants

`kSecTrustSettingsDefaultRootCertSetting`
Default trust settings for root certificates.

Available in Mac OS X v10.5 and later.

Declared in `SecTrustSettings.h`.

Discussion

Use this value with the `SecTrustSettingsSetTrustSettings` (page 80) function to set the default trust settings for root certificates. When evaluating trust settings for a root certificate in a given domain, if no matching explicit trust settings exist for that certificate, then the default value for the effective trust setting is returned (assuming that a default has been set and that the result is not `kSecTrustSettingsResultUnspecified`).

Result Codes

The most common result codes returned by Certificate, Key, and Trust Services are listed in the table below. The assigned error space is discontinuous: –25240..–25279 and –25290..–25329.

Result Code	Value	Description
`errSecNotAvailable`	–25291	No keychain is available. Available in Mac OS X v10.2 and later.
`errSecReadOnly`	–25292	A read-only error occurred. Available in Mac OS X v10.2 and later.
`errSecAuthFailed`	–25293	Authorization or authentication failed. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSecNoSuchKeychain	–25294	The keychain does not exist. Available in Mac OS X v10.2 and later.
errSecInvalidKeychain	–25295	The keychain is not valid. Available in Mac OS X v10.2 and later.
errSecDuplicateKeychain	–25296	A keychain with the same name already exists. Available in Mac OS X v10.2 and later.
errSecDuplicateItem	–25299	An item with the same primary key attributes already exists. Available in Mac OS X v10.2 and later.
errSecItemNotFound	–25300	The item cannot be found. Available in Mac OS X v10.2 and later.
errSecBufferTooSmall	–25301	The buffer is too small. Available in Mac OS X v10.2 and later.
errSecDataTooLarge	–25302	The data is too large for the particular data type. Available in Mac OS X v10.2 and later.
errSecNoSuchAttr	–25303	The attribute does not exist. Available in Mac OS X v10.2 and later.
errSecInvalidItemRef	–25304	The item object is invalid. Available in Mac OS X v10.2 and later.
errSecInvalidSearchRef	–25305	The search object is invalid. Available in Mac OS X v10.2 and later.
errSecNoSuchClass	–25306	The specified item does not appear to be a valid keychain item. Available in Mac OS X v10.2 and later.
errSecNoDefaultKeychain	–25307	A default keychain does not exist. Available in Mac OS X v10.2 and later.
errSecInteractionNotAllowed	–25308	Interaction with the user is required in order to grant access or process a request; however, user interaction with the Security Server has been disabled by the program. Available in Mac OS X v10.2 and later.
errSecReadOnlyAttr	–25309	The attribute is read-only. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSecWrongSecVersion	–25310	The version is incorrect. Available in Mac OS X v10.2 and later.
errSecKeySizeNotAllowed	–25311	The key size is not allowed. Available in Mac OS X v10.2 and later.
errSecNoStorageModule	–25312	No storage module is available. Available in Mac OS X v10.2 and later.
errSecNoCertificateModule	–25313	No certificate module is available. Available in Mac OS X v10.2 and later.
errSecNoPolicyModule	–25314	No policy module is available. Available in Mac OS X v10.2 and later.
errSecInteractionRequired	–25315	Interaction with the user is required in order to grant access or process a request; however, user interaction with the Security Server is impossible because the program is operating in a session incapable of graphics (such as a root session or ssh session). Available in Mac OS X v10.2 and later.
errSecDataNotAvailable	–25316	The data is not available. Available in Mac OS X v10.2 and later.
errSecDataNotModifiable	–25317	The data is not modifiable. Available in Mac OS X v10.2 and later.
errSecCreateChainFailed	–25318	One or more certificates required in order to validate this certificate cannot be found. Available in Mac OS X v10.2 and later.
errSecInvalidPrefsDomain	–25319	The preference domain specified is invalid. This error can occur in Mac OS X v10.3 and later. Available in Mac OS X v10.3 and later.
errSecACLNotSimple	–25240	The access control list is not in standard simple form. Available in Mac OS X v10.2 and later.
errSecPolicyNotFound	–25241	The policy specified cannot be found. Available in Mac OS X v10.2 and later.
errSecInvalidTrustSetting	–25242	The trust setting is invalid. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
`errSecNoAccessForItem`	–25243	The specified item has no access control. Available in Mac OS X v10.2 and later.
`errSecInvalidOwnerEdit`	–25244	An invalid attempt has been made to change the owner of an item. Available in Mac OS X v10.2 and later.
`errSecTrustNotAvailable`	–25245	No trust results are available. Available in Mac OS X v10.3 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Keychain Services Reference

Framework: Security/Security.h

Declared in SecItem.h
SecAccess.h
SecACL.h
SecBase.h
SecKeychain.h
SecKeychainItem.h
SecKeychainSearch.h

Overview

Keychain Services is a programming interface that enables you to find, add, modify, and delete keychain items.

Functions by Task

Getting Information About Keychain Services and Types

`SecKeychainGetVersion` (page 130)
Determines the version of Keychain Services installed on the user's system.

`SecKeychainGetTypeID` (page 129)
Returns the unique identifier of the opaque type to which a `SecKeychainRef` object belongs.

`SecKeychainItemGetTypeID` (page 139)
Returns the unique identifier of the opaque type to which a `SecKeychainItemRef` object belongs.

`SecKeychainSearchGetTypeID` (page 148)
Returns the unique identifier of the opaque type to which a `SecKeychainSearchRef` object belongs.

`SecAccessGetTypeID` (page 108)
Returns the unique identifier of the opaque type to which a `SecAccessRef` object belongs.

`SecACLGetTypeID` (page 111)
Returns the unique identifier of the opaque type to which a `SecACLRef` object belongs.

`SecTrustedApplicationGetTypeID` (page 155)
Returns the unique identifier of the opaque type to which a `SecTrustedApplication` object belongs.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Creating and Deleting a Keychain

`SecKeychainCreate` (page 121)
Creates an empty keychain.

`SecKeychainDelete` (page 123)
Deletes one or more keychains from the default keychain search list, and removes the keychain itself if it is a file.

Managing Keychains

`SecKeychainOpen` (page 145)
Opens a keychain.

`SecKeychainSetDefault` (page 149)
Sets the default keychain.

`SecKeychainCopyDefault` (page 119)
Retrieves a pointer to the default keychain.

`SecKeychainGetStatus` (page 129)
Retrieves status information of a keychain.

`SecKeychainGetPath` (page 127)
Determines the path of a keychain.

`SecKeychainSetSettings` (page 151)
Changes the settings of a keychain.

`SecKeychainCopySettings` (page 121)
Obtains a keychain's settings.

Locking and Unlocking Keychains

`SecKeychainLock` (page 144)
Locks a keychain.

`SecKeychainLockAll` (page 145)
Locks all keychains belonging to the current user.

`SecKeychainUnlock` (page 153)
Unlocks a keychain.

Managing User Interaction

`SecKeychainSetUserInteractionAllowed` (page 152)
Enables or disables the user interface for Keychain Services functions that automatically display a user interface.

`SecKeychainGetUserInteractionAllowed` (page 130)
Indicates whether Keychain Services functions that normally display a user interaction are allowed to do so.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Managing Keychain Access

`SecKeychainSetAccess` (page 148)
Sets the application access for a keychain.

`SecKeychainCopyAccess` (page 118)
Retrieves the application access of a keychain.

Storing and Retrieving Passwords

`SecKeychainAddInternetPassword` (page 116)
Adds a new Internet password to a keychain.

`SecKeychainFindInternetPassword` (page 124)
Finds the first Internet password based on the attributes passed.

`SecKeychainAddGenericPassword` (page 114)
Adds a new generic password to a keychain.

`SecKeychainFindGenericPassword` (page 123)
Finds the first generic password based on the attributes passed.

Searching for Keychain Items

`SecKeychainSetSearchList` (page 151)
Specifies the list of keychains to use in the default keychain search list.

`SecKeychainCopySearchList` (page 120)
Retrieves a keychain search list.

`SecKeychainSearchCreateFromAttributes` (page 147)
Creates a search object matching a list of zero or more attributes.

`SecKeychainSearchCopyNext` (page 146)
Finds the next keychain item matching the given search criteria.

Creating and Deleting Keychain Items

`SecKeychainItemCreateFromContent` (page 135)
Creates a new keychain item from the supplied parameters.

`SecKeychainItemCreateCopy` (page 134)
Copies a keychain item from one keychain to another.

`SecKeychainItemDelete` (page 136)
Deletes a keychain item from the default keychain's permanent data store.

Exporting and Importing Keychain Items

`SecKeychainItemExport` (page 136)
Exports one or more certificates, keys, or identities.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SecKeychainItemImport` (page 140)
Imports one or more certificates, keys, or identities and adds them to a keychain.

Managing Keychain Items

`SecKeychainItemCopyAttributesAndData` (page 131)
Retrieves the data and/or attributes stored in the given keychain item.

`SecKeychainItemModifyAttributesAndData` (page 142)
Updates an existing keychain item after changing its attributes or data.

`SecKeychainItemFreeAttributesAndData` (page 137)
Releases the memory used by the keychain attribute list and/or the keychain data retrieved in a call to `SecKeychainItemCopyAttributesAndData`.

`SecKeychainItemCopyContent` (page 132)
Copies the data and attributes stored in the given keychain item.

`SecKeychainItemModifyContent` (page 142)
Updates an existing keychain item after changing its attributes and/or data.

`SecKeychainItemFreeContent` (page 138)
Releases the memory used by the keychain attribute list and/or the keychain data retrieved in a call to the `SecKeychainItemCopyContent` (page 132) function.

`SecKeychainAttributeInfoForItemID` (page 117)
Obtains tags for all possible attributes of a given item class.

`SecKeychainFreeAttributeInfo` (page 126)
Releases the memory acquired by calling the `SecKeychainAttributeInfoForItemID` function.

`SecKeychainItemCopyKeychain` (page 133)
Returns the keychain object of a given keychain item.

`SecKeychainItemSetAccess` (page 143)
Sets the access of a given keychain item.

`SecKeychainItemCopyAccess` (page 131)
Copies the access of a given keychain item.

Creating an Access Object

`SecAccessCreate` (page 105)
Creates a new access object.

`SecAccessCreateFromOwnerAndACL` (page 106)
Creates a new access object using the owner and access control list you provide.

Managing Access Objects

`SecAccessCopyACLList` (page 104)
Retrieves all the access control list entries of a given access object.

`SecAccessCopySelectedACLList` (page 105)
Retrieves selected access control lists from a given access object.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SecAccessGetOwnerAndACL` (page 107)
Retrieves the owner and the access control list of a given access object.

Managing Access Control List Objects

`SecACLCreateFromSimpleContents` (page 109)
Creates a new access control list entry from the application list, description, and prompt selector provided and adds it to an item's access object.

`SecACLRemove` (page 111)
Removes the specified access control list entry.

`SecACLCopySimpleContents` (page 108)
Returns the application list, description, and CSSM prompt selector for a given access control list entry.

`SecACLSetSimpleContents` (page 113)
Sets the application list, description, and prompt selector for a given access control list entry.

`SecACLGetAuthorizations` (page 110)
Retrieves the CSSM authorization tags of a given access control list entry.

`SecACLSetAuthorizations` (page 112)
Sets the CSSM authorization tags for a given access control list entry.

Managing Trusted Applications

`SecTrustedApplicationCopyData` (page 154)
Retrieves the data of a trusted application object.

`SecTrustedApplicationCreateFromPath` (page 154)
Creates a trusted application object based on the application specified by path.

`SecTrustedApplicationSetData` (page 155)
Sets the data of a given trusted application object.

Managing Preference Domains

`SecKeychainGetPreferenceDomain` (page 128)
Gets the current keychain preference domain.

`SecKeychainSetPreferenceDomain` (page 150)
Sets the keychain preference domain.

`SecKeychainCopyDomainDefault` (page 119)
Retrieves the default keychain from a specified preference domain.

`SecKeychainSetDomainDefault` (page 149)
Sets the default keychain for a specified preference domain.

`SecKeychainCopyDomainSearchList` (page 120)
Retrieves the keychain search list for a specified preference domain.

`SecKeychainSetDomainSearchList` (page 150)
Sets the keychain search list for a specified preference domain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

CSSM Bridge Functions

`SecKeychainGetCSPHandle` (page 127)
Returns the CSSM CSP handle for the given keychain object.

`SecKeychainGetDLDBHandle` (page 127)
Returns the CSSM database handle for a given keychain object.

`SecKeychainItemGetDLDBHandle` (page 139)
Returns the CSSM database handle for a given keychain item object.

`SecKeychainItemGetUniqueRecordID` (page 139)
Returns a CSSM unique record for the given keychain item object.

Adding and Removing Callbacks

`SecKeychainAddCallback` (page 114)
Registers your keychain event callback function

`SecKeychainRemoveCallback` (page 146)
Unregisters your keychain event callback function.

Functions

SecAccessCopyACLList

Retrieves all the access control list entries of a given access object.

```
OSStatus SecAccessCopyACLList (
   SecAccessRef accessRef,
   CFArrayRef *aclList
);
```

Parameters

accessRef
The access object from which to retrieve the information.

aclList
On return, a pointer to a reference of a newly created `CFArray` of `SecACLRef` instances. You should call the `CFRelease` function on this array when you are finished with it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
An access object can have any number of access control list (ACL) entries for specific operations or sets of operations. To retrieve ACL entries for specific operations, use the `SecAccessCopySelectedACLList` (page 105) function.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecAccess.h

SecAccessCopySelectedACLList

Retrieves selected access control lists from a given access object.

```
OSStatus SecAccessCopySelectedACLList (
   SecAccessRef accessRef,
   CSSM_ACL_AUTHORIZATION_TAG action,
   CFArrayRef *aclList
);
```

Parameters

accessRef
The access object from which to retrieve the information.

action
An access control list authorization tag; the function returns only those access control list entries that apply to the operation indicated by this tag.

aclList
On return, a pointer to the selected access control lists. Release this by calling the `CFRelease` function.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
An access object can have any number of access control list (ACL) entries for specific operations or sets of operations. To retrieve all the ACL entries for an access object, use the `SecAccessCopyACLList` (page 104) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecAccess.h

SecAccessCreate

Creates a new access object.

```
OSStatus SecAccessCreate (
   CFStringRef descriptor,
   CFArrayRef trustedlist,
   SecAccessRef *accessRef
);
```

Parameters

descriptor
A `CFString` object representing the name of the keychain item as it should appear in security dialogs. Note that this is not necessarily the same name as appears for that item in the Keychain Access application.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

trustedlist

A reference to an array of trusted application objects (values of type `SecTrustedApplicationRef`) specifying which applications should be allowed to access the item without triggering confirmation dialogs. Use the `SecTrustedApplicationCreateFromPath` (page 154) function to create trusted application objects. If you pass `NULL` for this parameter, the access control list is automatically set to the application creating the item. To set no applications, pass a `CFArrayRef` with no elements.

accessRef

On return, points to the new access object. Release this object by calling the `CFRelease` function when you no longer need it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Each protected keychain item (such as a password or private key) has an associated access object. The access object contains access control list (ACL) entries, which specify trusted applications and the operations for which those operations are trusted. When an application attempts to access a keychain item for a particular purpose (such as to sign a document), the system determines whether that application is trusted to access the item for that purpose. If it is trusted, then the application is given access and no user confirmation is required. If the application is not trusted, but there is an ACL entry for that operation, then the user is asked to confirm whether the application should be given access. If there is no ACL entry for that operation, then access is denied and it is up to the calling application to try something else or to notify the user.

This function creates an access object with three ACL entries: The first, referred to as *owner access*, determines who can modify the access object itself. By default, there are no trusted applications for owner access; the user is always prompted for permission if someone tries to change access controls. The second is for operations considered safe, such as encrypting data. This ACL entry applies to all applications. The third ACL entry is for operations that should be restricted, such as decrypting, signing, deriving keys, and exporting keys. This ACL entry applies to the trusted applications listed in the `trustedlist` parameter.

To retrieve all the ACL entries of an access object, use the `SecAccessCopyACLList` (page 104) function. To retrieve specific ACL entries, use the `SecAccessCopySelectedACLList` (page 105) function. To create a new ACL entry and add it to an access object, use `SecACLCreateFromSimpleContents` (page 109). To modify an existing ACL entry, use `SecACLSetSimpleContents` (page 113). To modify the operations for which an ACL entry is used, call the `SecACLSetAuthorizations` (page 112) function.

Because an ACL object is always associated with an access object, when you modify an ACL entry, you are modifying the access object as well. Therefore, there is no need for a separate function to write a modified ACL object back into the access object.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecAccess.h`

SecAccessCreateFromOwnerAndACL

Creates a new access object using the owner and access control list you provide.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecAccessCreateFromOwnerAndACL (
   const CSSM_ACL_OWNER_PROTOTYPE *owner,
   uint32 aclCount,
   const CSSM_ACL_ENTRY_INFO *acls,
   SecAccessRef *accessRef
);
```

Parameters

owner

A pointer to a CSSM access control list owner.

aclCount

An unsigned 32-bit integer representing the number of items in the access control list.

acls

A pointer to the CSSM access control list.

accessRef

On return, points to the new access object. Release this by calling the `CFRelease` function.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

This function creates an access object from CSSM structures. You can use this function to create an access object for use with other Certificate, Key, and Trust API functions if you want to use CSSM to create the access control list. CSSM allows more complex access controls than you can construct with the Certificate, Key, and Trust API. For more information about the CSSM API, see *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecAccess.h`

SecAccessGetOwnerAndACL

Retrieves the owner and the access control list of a given access object.

```
OSStatus SecAccessGetOwnerAndACL (
   SecAccessRef accessRef,
   CSSM_ACL_OWNER_PROTOTYPE_PTR *owner,
   uint32 *aclCount,
   CSSM_ACL_ENTRY_INFO_PTR *acls
);
```

Parameters

accessRef

An access object from which to retrieve the owner and access control list.

owner

On return, a pointer to a CSSM access control list owner.

aclCount

On return, a pointer to an unsigned 32-bit integer representing the number of items in the access control list.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

acls

On return, a pointer to the CSSM access control list.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This function returns CSSM structures for use with CSSM API functions.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecAccess.h`

SecAccessGetTypeID

Returns the unique identifier of the opaque type to which a `SecAccessRef` object belongs.

```
CFTypeID SecAccessGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecAccessRef` (page 157) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecAccessRef` (page 157) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecAccess.h`

SecACLCopySimpleContents

Returns the application list, description, and CSSM prompt selector for a given access control list entry.

```
OSStatus SecACLCopySimpleContents (
   SecACLRef acl,
   CFArrayRef *applicationList,
   CFStringRef *description,
   CSSM_ACL_KEYCHAIN_PROMPT_SELECTOR *promptSelector
);
```

Parameters

acl

An ACL object that identifies the access control list entry from which you want information.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

applicationList

On return, points to an array of `SecTrustedApplication` instances identifying applications that are allowed access to the keychain item without user confirmation. If this parameter returns `NULL`, then any application can use this item. If this parameter returns a valid pointer but the array is empty, then there are no trusted applications. Call `CFRelease` for this object when you no longer need it.

description

On return, the name of the keychain item that appears in the dialog box when the user is prompted for permission to use the item. Note that this name is not necessarily the same as the one displayed for the item by the Keychain Access application. Call `CFRelease` for this object when you no longer need it.

promptSelector

On return, points to the prompt selector flag for the given access control list entry. If the `CSSM_ACL_KEYCHAIN_PROMPT_REQUIRE_PASSPHRASE` bit is set, the user is prompted for the keychain password each time a non-trusted application attempts to access this item, even if the keychain is already unlocked.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
An access control list entry applies to a specific use or set of uses for a specific keychain item. The ACL object includes a list of trusted applications (see `SecTrustedApplicationCreateFromPath` (page 154)), the name of the keychain item as it appears in user prompts, the prompt selector flag, and a list of one or more operations to which this ACL object applies. Use the `SecACLGetAuthorizations` (page 110) function to get the list of operations for an ACL object.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecACL.h`

SecACLCreateFromSimpleContents

Creates a new access control list entry from the application list, description, and prompt selector provided and adds it to an item's access object.

```
OSStatus SecACLCreateFromSimpleContents (
   SecAccessRef access,
   CFArrayRef applicationList,
   CFStringRef description,
   const CSSM_ACL_KEYCHAIN_PROMPT_SELECTOR *promptSelector,
   SecACLRef *newAcl
);
```

Parameters

access

The access object to which to add the information.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

applicationList

An array of trusted application objects (that is, `SecTrustedApplication` instances) identifying applications that are allowed access to the keychain item without user confirmation. Use the `SecTrustedApplicationCreateFromPath` (page 154) function to create trusted application objects. If you set this parameter to `NULL`, then any application can use this item. If you pass an empty array, then there are no trusted applications. Call `CFRelease` for this object when you no longer need it.

description

The human readable name to be used to refer to this item when the user is prompted.

promptSelector

A pointer to a prompt selector. If you set the `CSSM_ACL_KEYCHAIN_PROMPT_REQUIRE_PASSPHRASE` bit, the user is prompted for the keychain password each time a non-trusted application attempts to access this item, even if the keychain is already unlocked.

newAcl

On return, points to an access control list object, which is a reference to the new access control list entry.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
The ACL object returned by this function is a reference to an access control list (ACL) entry. The ACL entry includes a list of trusted applications (see `SecTrustedApplicationCreateFromPath` (page 154)), the name of the keychain item as it appears in user prompts, the prompt selector flag, and a list of one or more operations to which this ACL entry applies. By default, a new ACL entry applies to all operations (the CSSM authorization tag is set to `CSSM_ACL_AUTHORIZATION_ANY`). Use the `SecACLSetAuthorizations` (page 112) function to set the list of operations for an ACL object.

The system allows exactly one owner ACL entry in each access object. The `SecACLCreateFromSimpleContents` function fails if you attempt to add a second owner ACL. To change owner access controls, use the `SecAccessCopySelectedACLList` (page 105) function to find the owner ACL (that is, the only ACL with a CSSM authorization tag of `CSSM_ACL_AUTHORIZATION_CHANGE_ACL`) and the `SecACLSetSimpleContents` (page 113) function to change it as needed.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecACL.h`

SecACLGetAuthorizations

Retrieves the CSSM authorization tags of a given access control list entry.

```
OSStatus SecACLGetAuthorizations (
   SecACLRef acl,
   CSSM_ACL_AUTHORIZATION_TAG *tags,
   uint32 *tagCount
);
```

Parameters

acl

An ACL object that identifies the access control list entry from which you wish to retrieve the authorization tags.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

tags

A pointer to an array of CSSM authorization tags. You must allocate this array before calling the function. On return, this array contains the authorization tags of the specified ACL entry.

tagCount

On input, points to the number of elements in the array you passed in the `tags` parameter. On return, points to the number of tags actually returned.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
An ACL object includes a list of trusted applications (see `SecTrustedApplicationCreateFromPath` (page 154)), the name of the keychain item as it appears in user prompts, the prompt selector flag, and a list of one or more operations to which this ACL object applies. Use this function to retrieve the list of operations for an ACL object. Use the `SecACLCopySimpleContents` (page 108) function to retrieve the other information.

The `SecACLGetAuthorizations` function returns an error if there are more tags to return than the number of elements you allocated in the `tags` array. A 20-element array should suffice for most purposes; however, you can test for the `errSecBufferTooSmall` error and increase the size of the array before calling the function again if necessary.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecACL.h`

SecACLGetTypeID

Returns the unique identifier of the opaque type to which a `SecACLRef` object belongs.

```
CFTypeID SecACLGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecACLRef` (page 157) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecACLRef` (page 157) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecACL.h`

SecACLRemove

Removes the specified access control list entry.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecACLRemove (
   SecACLRef aclRef
);
```

Parameters

aclRef

An ACL object that identifies the access control list entry to remove.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
The system allows exactly one owner ACL entry in each access object. The `SecACLRemove` function fails if you attempt to remove the owner ACL entry. To change owner access controls, use the `SecAccessCopySelectedACLList` (page 105) function to find the owner ACL (that is, the only ACL with a CSSM authorization tag of `CSSM_ACL_AUTHORIZATION_CHANGE_ACL`) and the `SecACLSetSimpleContents` (page 113) function to change it as needed.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecACL.h`

SecACLSetAuthorizations

Sets the CSSM authorization tags for a given access control list entry.

```
OSStatus SecACLSetAuthorizations (
   SecACLRef acl,
   CSSM_ACL_AUTHORIZATION_TAG *tags,
   uint32 tagCount
);
```

Parameters

acl

An ACL object that identifies the access control list entry for which you wish to set authorization tags.

tags

An array of CSSM authorization tags.

tagCount

The number of tags in the CSSM authorization tag array.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
An ACL object includes a list of trusted applications (see `SecTrustedApplicationCreateFromPath` (page 154)), the name of the keychain item as it appears in user prompts, the prompt selector flag, and a list of one or more operations to which this ACL object applies. Use this function to set a list of operations for an ACL object, or set the `CSSM_ACL_AUTHORIZATION_ANY` tag to allow all operations. Use the `SecACLSetSimpleContents` (page 113) function to set the other information.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Because an ACL object is always associated with an access object, when you modify an ACL entry, you are modifying the access object as well. There is no need for a separate function to write a modified ACL object back into the access object.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecACL.h`

SecACLSetSimpleContents

Sets the application list, description, and prompt selector for a given access control list entry.

```
OSStatus SecACLSetSimpleContents (
   SecACLRef acl,
   CFArrayRef applicationList,
   CFStringRef description,
   const CSSM_ACL_KEYCHAIN_PROMPT_SELECTOR *promptSelector
);
```

Parameters

acl

An ACL object that identifies the access control list entry.

applicationList

An array of trusted application objects (that is, `SecTrustedApplication` instances) identifying applications that are allowed access to the keychain item without user confirmation. Use the `SecTrustedApplicationCreateFromPath` (page 154) function to create trusted application objects. If you set this parameter to `NULL`, then any application can use this item. If you pass an empty array, then there are no trusted applications. Call `CFRelease` for this object when you no longer need it.

description

The name of the keychain item that appears in the dialog box when the user is prompted for permission to use the item. Note that this name is not necessarily the same as the one displayed for the item by the Keychain Access application. Call `CFRelease` for this object when you no longer need it.

promptSelector

The prompt selector flag for the given access control list entry. Set the `CSSM_ACL_KEYCHAIN_PROMPT_REQUIRE_PASSPHRASE` bit to have the user prompted for the keychain password each time a non-trusted application attempts to access this item, even if the keychain is already unlocked.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Because an ACL object is always associated with an access object, when you modify an ACL entry, you are modifying the access object as well. There is no need for a separate function to write a modified ACL object back into the access object.

Use the `SecACLGetAuthorizations` (page 110) function to get the list of operations for an ACL object.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecACL.h

SecKeychainAddCallback

Registers your keychain event callback function

```
OSStatus SecKeychainAddCallback (
   SecKeychainCallback callbackFunction,
   SecKeychainEventMask eventMask,
   void *userContext
);
```

Parameters

callbackFunction
A pointer to your keychain event callback function, described in SecKeychainCallback (page 156).

eventMask
A bit mask indicating the keychain events of which your application wishes to be notified. Keychain Services tests this mask to determine the keychain events that you wish to receive, and passes these events in the keychainEvent parameter of your callback function.

userContext
A pointer to application-defined storage that will be passed to your callback function. Your application can use this to associate any particular call of this function with any particular call of your keychain event callback function.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
It is important to note that the current Foundation or Core Foundation run loop must be active when making this call or the callbacks are not registered. In multithreaded programs, the notifications are registered in the run loop of the thread calling SecKeychainAddCallback; therefore, delivery of notifications depends on the functioning of that thread's run loop. If that thread terminates, or is so busy that it doesn't operate its run loop in a timely manner, notifications will be delayed, and may eventually be dropped without any notification.

For that reason, it is inadvisable for your program to depend on delivery of notifications caused by your own actions (such as depending on receiving a deletion notification before updating a UI view) unless your program is multithreaded and can take notifications on a thread different from the one generating the events.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

SecKeychainAddGenericPassword

Adds a new generic password to a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainAddGenericPassword (
   SecKeychainRef keychain,
   UInt32 serviceNameLength,
   const char *serviceName,
   UInt32 accountNameLength,
   const char *accountName,
   UInt32 passwordLength,
   const void *passwordData,
   SecKeychainItemRef *itemRef
);
```

Parameters

keychain

A reference to the keychain in which to store a generic password. Pass `NULL` to specify the default keychain.

serviceNameLength

The length of the `serviceName` character string.

serviceName

A UTF-8 encoded character string representing the service name.

accountNameLength

The length of the `accountName` character string.

accountName

A UTF-8 encoded character string representing the account name.

passwordLength

The length of the `passwordData` buffer.

passwordData

A pointer to a buffer containing the password data to be stored in the keychain. Before calling this function, allocate enough memory for the buffer to hold the data you want to store.

itemRef

On return, a pointer to a reference to the new keychain item. Pass `NULL` if you don't want to obtain this object. You must allocate the memory for this pointer. The memory that this pointer occupies must be released by calling the `CFRelease` function when finished with it.

Return Value

A result code. See "Keychain Services Result Codes" (page 191). The result code `errSecNoDefaultKeychain` indicates that no default keychain could be found. The result code `errSecDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errSecDataTooLarge` indicates that you tried to add more data than is allowed for a structure of this type.

Discussion

This function adds a new generic password to the specified keychain. Required parameters to identify the password are `serviceName` and `accountName`, which are application-defined strings. This function optionally returns a reference to the newly added item.

You can use this function to add passwords for accounts other than the Internet. For example, you might add AppleShare passwords, or passwords for your database or scheduling programs.

This function sets the initial access rights for the new keychain item so that the application creating the item is given trusted access.

This function automatically calls the function `SecKeychainUnlock` (page 153) to display the Unlock Keychain dialog box if the keychain is currently locked.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainAddInternetPassword

Adds a new Internet password to a keychain.

```
OSStatus SecKeychainAddInternetPassword (
   SecKeychainRef keychain,
   UInt32 serverNameLength,
   const char *serverName,
   UInt32 securityDomainLength,
   const char *securityDomain,
   UInt32 accountNameLength,
   const char *accountName,
   UInt32 pathLength,
   const char *path,
   UInt16 port,
   SecProtocolType protocol,
   SecAuthenticationType authenticationType,
   UInt32 passwordLength,
   const void *passwordData,
   SecKeychainItemRef *itemRef
);
```

Parameters

keychain
A reference to the keychain in which to store an Internet password. Pass `NULL` to specify the user's default keychain.

serverNameLength
The length of the `serverName` character string.

serverName
A UTF-8 encoded character string representing the server name.

securityDomainLength
The length of the `securityDomain` character string.

securityDomain
A UTF-8 encoded character string representing the security domain. This parameter is optional. Pass NULL if the protocol does not require it.

accountNameLength
The length of the `accountName` character string.

accountName
A UTF-8 encoded character string representing the account name.

pathLength
The length of the `path` character string.

path
A UTF-8 encoded character string representing the path.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

port
The TCP/IP port number. If no specific port number is associated with this password, pass `0`.

protocol
The protocol associated with this password. See "Keychain Protocol Type Constants" (page 186) for a description of possible values.

authenticationType
The authentication scheme used. See "Keychain Authentication Type Constants" (page 166) for a description of possible values. Pass the constant `kSecAuthenticationTypeDefault`, to specify the default authentication scheme.

passwordLength
The length of the `passwordData` buffer.

passwordData
A pointer to a buffer containing the password data to be stored in the keychain.

itemRef
On return, a pointer to a reference to the new keychain item. Pass `NULL` if you don't want to obtain this object. You must allocate the memory for this pointer. The memory that this pointer occupies must be released by calling the `CFRelease` function when finished with it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191). The result code `errSecNoDefaultKeychain` indicates that no default keychain could be found. The result code `errSecDuplicateItem` indicates that you tried to add a password that already exists in the keychain. The result code `errSecDataTooLarge` indicates that you tried to add more data than is allowed for a structure of this type.

Discussion
This function adds a new Internet server password to the specified keychain. Required parameters to identify the password are `serverName` and `accountName` (you cannot pass `NULL` for both parameters). In addition, some protocols may require an optional `securityDomain` when authentication is requested. This function optionally returns a reference to the newly added item.

This function sets the initial access rights for the new keychain item so that the application creating the item is given trusted access.

This function automatically calls the function `SecKeychainUnlock` (page 153) to display the Unlock Keychain dialog box if the keychain is currently locked.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
ImageClient

Declared In
`SecKeychain.h`

SecKeychainAttributeInfoForItemID

Obtains tags for all possible attributes of a given item class.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainAttributeInfoForItemID (
   SecKeychainRef keychain,
   UInt32 itemID,
   SecKeychainAttributeInfo **info
);
```

Parameters

keychain

A keychain object.

itemID

The relation identifier of the item tags. An *itemID* is a `CSSM_DB_RECORDTYPE` type as defined in `cssmtype.h`.

info

On return, a pointer to the keychain attribute information. Your application should call the `SecKeychainFreeAttributeInfo` (page 126) function to release this structure when done with it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This call returns more attributes than are supported by the old style Keychain API and passing them into older calls yields an invalid attribute error. The recommended call to retrieve the attribute values is the `SecKeychainItemCopyAttributesAndData` (page 131) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainCopyAccess

Retrieves the application access of a keychain.

```
OSStatus SecKeychainCopyAccess (
   SecKeychainRef keychain,
   SecAccessRef *access
);
```

Parameters

keychain

A reference to the keychain from which to copy the access object. Pass `NULL` to specify the default keychain.

access

A pointer to an access object. On return, this points to the access object of the specified keychain. See "Managing Access Objects" (page 102) for information on manipulating access objects.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Special Considerations

Although this function is available in Mac OS X v10.2, it was unimplemented before Mac OS X v10.3 and returned an `unimpErr` error code if called.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainCopyDefault

Retrieves a pointer to the default keychain.

```
OSStatus SecKeychainCopyDefault (
   SecKeychainRef *keychain
);
```

Parameters

keychain

On return, a pointer to the default keychain object. The memory that this pointer occupies must be released by calling the `CFRelease` function when finished with it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191). The result code `errSecNoDefaultKeychain` indicates that there is no default keychain.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecKeychain.h`

SecKeychainCopyDomainDefault

Retrieves the default keychain from a specified preference domain.

```
OSStatus SecKeychainCopyDomainDefault (
   SecPreferencesDomain domain,
   SecKeychainRef *keychain
);
```

Parameters

domain

The preference domain from which you wish to retrieve the default keychain. See "Keychain Preference Domain Constants" (page 185) for possible domain values.

keychain

On return, a pointer to the keychain object of the default keychain in the specified preference domain.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. Use this function if you want to retrieve the default keychain for a specific preference domain. Use the `SecKeychainCopyDefault` (page 119) function if you want the default keychain for the current preference domain. See the `SecKeychainSetPreferenceDomain` (page 150) function for a discussion of current and default preference domains.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecKeychain.h`

SecKeychainCopyDomainSearchList

Retrieves the keychain search list for a specified preference domain.

```
OSStatus SecKeychainCopyDomainSearchList (
   SecPreferencesDomain domain,
   CFArrayRef *searchList
);
```

Parameters

domain

The preference domain from which you wish to retrieve the keychain search list. See "Keychain Preference Domain Constants" (page 185) for possible domain values.

searchList

On return, a pointer to the keychain search list of the specified preference domain.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. Use this function if you want to retrieve the keychain search list for a specific preference domain. Use the `SecKeychainCopySearchList` (page 120) function if you want the keychain search list for the current preference domain. See the `SecKeychainSetPreferenceDomain` (page 150) function for a discussion of current and default preference domains.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecKeychain.h`

SecKeychainCopySearchList

Retrieves a keychain search list.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainCopySearchList (
   CFArrayRef *searchList
);
```

Parameters

searchList

The returned keychain search list. When finished with the array, you must call `CFRelease` to release the memory.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychain.h`

SecKeychainCopySettings

Obtains a keychain's settings.

```
OSStatus SecKeychainCopySettings (
   SecKeychainRef keychain,
   SecKeychainSettings *outSettings
);
```

Parameters

keychain

A reference to the keychain from which to copy its settings.

outSettings

On return, a pointer to a keychain settings structure. Since this structure is versioned, you must allocate the memory for it and fill in the version of the structure before passing it to the function.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychain.h`

SecKeychainCreate

Creates an empty keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainCreate (
   const char *pathName,
   UInt32 passwordLength,
   const void *password,
   Boolean promptUser,
   SecAccessRef initialAccess,
   SecKeychainRef *keychain
);
```

Parameters

pathName

A constant character string representing the POSIX path indicating where to store the keychain.

passwordLength

An unsigned 32-bit integer representing the length of the buffer pointed to by `password`. Pass `0` if the value of `password` is `NULL` and the value of `promptUser` is `TRUE`.

password

A pointer to the buffer containing the password which is used to protect the new keychain. The password must be in canonical UTF8 encoding. Pass `NULL` if the value of `passwordLength` is `0` and the value of `promptUser` is `TRUE`.

promptUser

A Boolean value representing whether to display a password dialog to the user. Set this value to `TRUE` to display a password dialog or `FALSE` otherwise. If you pass `TRUE`, any values passed for `passwordLength` and `password` are ignored, and a dialog for the user to enter a password is presented.

initialAccess

An access object indicating the initial access rights for the keychain. A keychain's access rights determine which applications have permission to use the keychain. You may pass `NULL` for the standard access rights.

keychain

On return, a pointer to a keychain object. The memory that the keychain object pointer occupies must be released by calling `CFRelease` when you are finished with it. Pass `NULL` if you do not need the pointer to the keychain object returned.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

This function creates an empty keychain. The `keychain`, `password`, and *`initialAccess`* parameters are optional. If user interaction to create a keychain is posted, the newly-created keychain is automatically unlocked after creation.

The system ensures that a default keychain is created for the user at login, thus, in most cases, you do not need to call this function yourself. Users can create additional keychains, or change the default, by using the Keychain Access application. However, a missing default keychain is not recreated automatically, and you may receive an `errSecNoDefaultKeychain` error from other functions if a default keychain does not exist. In that case, you can use this function followed by `SecKeychainSetDefault` (page 149), to create a new default keychain. You can also call this function to create a private temporary keychain for your application's use, in cases where no user interaction can occur.

Availability

Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecKeychain.h

SecKeychainDelete

Deletes one or more keychains from the default keychain search list, and removes the keychain itself if it is a file.

```
OSStatus SecKeychainDelete (
   SecKeychainRef keychainOrArray
);
```

Parameters

keychainOrArray

A single keychain object or a reference to an array of keychains you wish to delete. To delete more than one keychain, create a `CFArray` of keychain references (type `SecKeychainRef`) and pass a reference to the array. In Mac OS X v10.3 and later, passing `NULL` to this parameter returns an `errSecInvalidKeychain` error code.

In Mac OS X v10.2, this parameter was named `keychain` and only took a single keychain object. Passing `NULL` to this parameter deleted the user's default keychain.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
The keychain may be a file stored locally, a smart card, or retrieved from a network server using non-file-based database protocols. This function deletes the keychain only if it is a local file.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

SecKeychainFindGenericPassword

Finds the first generic password based on the attributes passed.

```
OSStatus SecKeychainFindGenericPassword (
   CFTypeRef keychainOrArray,
   UInt32 serviceNameLength,
   const char *serviceName,
   UInt32 accountNameLength,
   const char *accountName,
   UInt32 *passwordLength,
   void **passwordData,
   SecKeychainItemRef *itemRef
);
```

Parameters

keychainOrArray

A reference to an array of keychains to search, a single keychain, or `NULL` to search the user's default keychain search list.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

serviceNameLength
The length of the `serviceName` character string.

serviceName
A UTF-8 encoded character string representing the service name.

accountNameLength
The length of the `accountName` character string.

accountName
A UTF-8 encoded character string representing the account name.

passwordLength
On return, the length of the buffer pointed to by `passwordData`.

passwordData
On return, a pointer to a buffer that holds the password data. Pass `NULL` if you want to obtain the item object but not the password data. In this case, you must also pass `NULL` in the `passwordLength` parameter. You should use the `SecKeychainItemFreeContent` (page 138) function to free the memory pointed to by this parameter.

itemRef
On return, a pointer to the item object of the generic password. Pass `NULL` if you don't want to obtain this object.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This function finds the first generic password item that matches the attributes you provide. Most attributes are optional; you should pass only as many as you need to narrow the search sufficiently for your application's intended use. This function optionally returns a reference to the found item.

This function decrypts the password before returning it to you. If the calling application is not in the list of trusted applications, the user is prompted before access is allowed. If the access controls for this item do not allow decryption, the function returns the `errSecAuthFailed` result code.

This function automatically calls the function `SecKeychainUnlock` (page 153) to display the Unlock Keychain dialog box if the keychain is currently locked.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainFindInternetPassword

Finds the first Internet password based on the attributes passed.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainFindInternetPassword (
   CFTypeRef keychainOrArray,
   UInt32 serverNameLength,
   const char *serverName,
   UInt32 securityDomainLength,
   const char *securityDomain,
   UInt32 accountNameLength,
   const char *accountName,
   UInt32 pathLength,
   const char *path,
   UInt16 port,
   SecProtocolType protocol,
   SecAuthenticationType authenticationType,
   UInt32 *passwordLength,
   void **passwordData,
   SecKeychainItemRef *itemRef
);
```

Parameters

keychainOrArray

A reference to an array of keychains to search, a single keychain or `NULL` to search the user's default keychain search list.

serverNameLength

The length of the `serverName` character string.

serverName

A UTF-8 encoded character string representing the server name.

securityDomainLength

The length of the `securityDomain` character string.

securityDomain

A UTF-8 encoded character string representing the security domain. This parameter is optional, as not all protocols require it. Pass `NULL` if it is not required.

accountNameLength

The length of the `accountName` character string.

accountName

A UTF-8 encoded character string representing the account name.

pathLength

The length of the `path` character string.

path

A UTF-8 encoded character string representing the path.

port

The TCP/IP port number. Pass `0` to ignore the port number.

protocol

The protocol associated with this password. See "Keychain Protocol Type Constants" (page 186) for a description of possible values.

authenticationType

The authentication scheme used. See "Keychain Authentication Type Constants" (page 166) for a description of possible values. Pass the constant `kSecAuthenticationTypeDefault`, to specify the default authentication scheme.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

passwordLength
On return, the length of the buffer pointed to by `passwordData`.

passwordData
On return, a pointer to a buffer containing the password data. Pass `NULL` if you want to obtain the item object but not the password data. In this case, you must also pass `NULL` in the `passwordLength` parameter. You should use the `SecKeychainItemFreeContent` (page 138) function to free the memory pointed to by this parameter.

itemRef
On return, a pointer to the item object of the Internet password. Pass `NULL` if you don't want to obtain this object.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This function finds the first Internet password item that matches the attributes you provide. This function optionally returns a reference to the found item.

This function decrypts the password before returning it to you. If the calling application is not in the list of trusted applications, the user is prompted before access is allowed. If the access controls for this item do not allow decryption, the function returns the `errSecAuthFailed` result code.

This function automatically calls the function `SecKeychainUnlock` (page 153) to display the Unlock Keychain dialog box if the keychain is currently locked.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
ImageClient

Declared In
`SecKeychain.h`

SecKeychainFreeAttributeInfo

Releases the memory acquired by calling the `SecKeychainAttributeInfoForItemID` function.

```
OSStatus SecKeychainFreeAttributeInfo (
   SecKeychainAttributeInfo *info
);
```

Parameters

info
A pointer to the keychain attribute information to release.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecKeychainGetCSPHandle

Returns the CSSM CSP handle for the given keychain object.

```
OSStatus SecKeychainGetCSPHandle (
   SecKeychainRef keychain,
   CSSM_CSP_HANDLE *cspHandle
);
```

Parameters

keychain

A keychain object.

cspHandle

On return, a pointer to the CSSM CSP handle for the given keychain. The handle is valid until the keychain object is released.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

Declared In

SecKeychain.h

SecKeychainGetDLDBHandle

Returns the CSSM database handle for a given keychain object.

```
OSStatus SecKeychainGetDLDBHandle (
   SecKeychainRef keychain,
   CSSM_DL_DB_HANDLE *dldbHandle
);
```

Parameters

keychain

A keychain object.

dldbHandle

On return, a pointer to the CSSM database handle for the given keychain. The handle is valid until the keychain object is released.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

Declared In

SecKeychain.h

SecKeychainGetPath

Determines the path of a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainGetPath (
   SecKeychainRef keychain,
   UInt32 *ioPathLength,
   char *pathName
);
```

Parameters

keychain

A reference to a keychain whose path you wish to obtain.

ioPathLength

On input, a pointer to the size of the character string `pathName`. On return, the size of `pathName` without the zero termination.

pathName

On input, a pointer to a buffer that you have allocated. On output, the buffer contains the POSIX path of the keychain as a UTF-8 encoded string. The function returns `errSecBufferTooSmall` if the provided buffer is too small.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecKeychain.h`

SecKeychainGetPreferenceDomain

Gets the current keychain preference domain.

```
OSStatus SecKeychainGetPreferenceDomain (
   SecPreferencesDomain *domain
);
```

Parameters

domain

On return, a pointer to the keychain preference domain. See "Keychain Preference Domain Constants" (page 185) for possible domain values.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. The default preference domain for system daemons (that is, for daemons running in the root session) is the system domain. The default preference domain for all other programs is the user domain. Use the `SecKeychainSetPreferenceDomain` (page 150) function to change the preference domain.

Availability
Available in Mac OS X v10.3 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecKeychain.h

SecKeychainGetStatus

Retrieves status information of a keychain.

```
OSStatus SecKeychainGetStatus (
   SecKeychainRef keychain,
   SecKeychainStatus *keychainStatus
);
```

Parameters

keychain

A keychain object of the keychain whose status you wish to determine for the user session. Pass NULL to obtain the status of the default keychain.

keychainStatus

On return, a pointer to the status of the specified keychain. See "Keychain Status Masks" (page 190) for valid status constants.

Return Value
A result code. See "Keychain Services Result Codes" (page 191). The result code errSecNoSuchKeychain indicates that the specified keychain could not be found. The result code errSecInvalidKeychain indicates that the specified keychain is invalid.

Discussion
This function retrieves the status of a specified keychain. You can use this function to determine if the keychain is unlocked, readable, or writable. Note that the lock status of a keychain can change at any time due to user or system activity. Because the system automatically prompts the user to unlock a keychain when necessary, you do not usually have to worry about the lock status of a keychain. If you do need to track the lock status of a keychain, use the SecKeychainAddCallback (page 114) function to register for keychain notifications.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

SecKeychainGetTypeID

Returns the unique identifier of the opaque type to which a SecKeychainRef object belongs.

```
CFTypeID SecKeychainGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a SecKeychainRef (page 160) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a SecKeychainRef (page 160) object. You can compare this value to the CFTypeID identifier obtained by calling the CFGetTypeID function on a specific object. These values might change from release to release or platform to platform.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

SecKeychainGetUserInteractionAllowed

Indicates whether Keychain Services functions that normally display a user interaction are allowed to do so.

```
OSStatus SecKeychainGetUserInteractionAllowed (
   Boolean *state
);
```

Parameters

state

A Boolean value indicating whether user interaction is permitted. If `true`, user interaction is allowed, and Keychain Services functions that display a user interface can do so as appropriate.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

SecKeychainGetVersion

Determines the version of Keychain Services installed on the user's system.

```
OSStatus SecKeychainGetVersion (
   UInt32 *returnVers
);
```

Parameters

returnVers

On return, a pointer to the version number of Keychain Services installed on the current system. See "Keychain Settings Version" (page 190) for a list of values.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Your application can call the `SecKeychainGetVersion` function to find out which version of Keychain Services is installed on the user's system.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecKeychainItemCopyAccess

Copies the access of a given keychain item.

```
OSStatus SecKeychainItemCopyAccess (
   SecKeychainItemRef itemRef,
   SecAccessRef *access
);
```

Parameters

itemRef

A reference to a keychain item.

access

On return, points to the keychain item's access object. Release this object by calling the `CFRelease` function.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

Each protected keychain item (such as a password or private key) has an associated access object. The access object contains access control list (ACL) entries, which specify trusted applications and the operations for which those operations are trusted. You can use this function together with the `SecKeychainItemSetAccess` (page 143) function to copy access controls from one keychain item to another. You can use the functions in the section "Managing Access Control List Objects" (page 103) to modify the contents of an access object.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychainItem.h`

SecKeychainItemCopyAttributesAndData

Retrieves the data and/or attributes stored in the given keychain item.

```
OSStatus SecKeychainItemCopyAttributesAndData (
   SecKeychainItemRef itemRef,
   SecKeychainAttributeInfo *info,
   SecItemClass *itemClass,
   SecKeychainAttributeList **attrList,
   UInt32 *length,
   void **outData
);
```

Parameters

itemRef

A reference to the keychain item from which you wish to retrieve data or attributes.

info

A pointer to a list of tags of attributes to retrieve.

itemClass

A pointer to the item's class. You should pass `NULL` if not required. See "Keychain Item Class Constants" (page 179) for valid constants.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

attrList

On input, the list of attributes in this item to get; on output the attributes are filled in. You should call the function `SecKeychainItemFreeAttributesAndData` (page 137) when you no longer need the attributes and data.

length

On return, a pointer to the actual length of the data.

outData

A pointer to a buffer containing the data in this item. Pass `NULL` if not required. You should call the function `SecKeychainItemFreeAttributesAndData` (page 137) when you no longer need the attributes and data.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This function returns the data and attributes of a specific keychain item. You can use the `SecKeychainSearchCopyNext` (page 146) function to search for a keychain item if you don't already have the item's reference object. To find and obtain data from a password keychain item, use the `SecKeychainFindInternetPassword` (page 124) or `SecKeychainFindGenericPassword` (page 123) function.

You should pair the `SecKeychainItemCopyAttributesAndData` function with the `SecKeychainItemModifyAttributesAndData` (page 142) function, as these functions handle more attributes than are support by the old Keychain Manager and passing them into older calls yields an invalid attribute error. Use the functions `SecKeychainItemModifyContent` (page 142) and `SecKeychainItemCopyContent` (page 132) when dealing with older Keychain Manager functions.

If the keychain item data is encrypted, this function decrypts the data before returning it to you. If the calling application is not in the list of trusted applications, the user is prompted before access is allowed. If the access controls for this item do not allow decryption, the function returns the `errSecAuthFailed` result code.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecKeychainItemCopyContent

Copies the data and attributes stored in the given keychain item.

```
OSStatus SecKeychainItemCopyContent (
   SecKeychainItemRef itemRef,
   SecItemClass *itemClass,
   SecKeychainAttributeList *attrList,
   UInt32 *length,
   void **outData
);
```

Parameters

itemRef

A reference to the keychain item to modify.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

itemClass
A pointer to the item's class. You should pass NULL if it is not required. See "Keychain Item Class Constants" (page 179) for valid constants.

attrList
A pointer to the list of attributes to get in this item on input; on output the attributes are filled in. You must call SecKeychainItemFreeContent (page 138) when you no longer need the attributes and data.

length
On return, the length of the buffer pointed to by the *outData* parameter.

outData
On return, a pointer to a buffer containing the data in this item. You must call SecKeychainItemFreeContent (page 138) when you no longer need the attributes and data.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This function returns the data and attributes of a specific keychain item. You can use the SecKeychainSearchCopyNext (page 146) function to search for a keychain item if you don't already have the item's reference object. To find and obtain data from a password keychain item, use the SecKeychainFindInternetPassword (page 124) or SecKeychainFindGenericPassword (page 123) function.

You should pair the SecKeychainItemModifyContent (page 142) function with the SecKeychainItemCopyContent function when dealing with older Keychain Manager functions. The SecKeychainItemCopyAttributesAndData (page 131) and SecKeychainItemModifyAttributesAndData (page 142) functions handle more attributes than are support by the old Keychain Manager; however, passing them into older calls yields an invalid attribute error.

If the keychain item data is encrypted, this function decrypts the data before returning it to you. If the calling application is not in the list of trusted applications, the user is prompted before access is allowed. If the access controls for this item do not allow decryption, the function returns the errSecAuthFailed result code.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
ImageClient

Declared In
SecKeychainItem.h

SecKeychainItemCopyKeychain

Returns the keychain object of a given keychain item.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainItemCopyKeychain (
   SecKeychainItemRef itemRef,
   SecKeychainRef *keychainRef
);
```

Parameters

itemRef

A keychain item object.

keychainRef

On return, a pointer to a keychain object referencing the given keychain item. Release this by calling the `CFRelease` function.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychainItem.h`

SecKeychainItemCreateCopy

Copies a keychain item from one keychain to another.

```
OSStatus SecKeychainItemCreateCopy (
   SecKeychainItemRef itemRef,
   SecKeychainRef destKeychainRef,
   SecAccessRef initialAccess,
   SecKeychainItemRef *itemCopy
);
```

Parameters

itemRef

A reference to the keychain item to copy.

destKeychainRef

A reference to the keychain in which to insert the copied keychain item. Pass `NULL` to specify the default keychain.

initialAccess

The initial access for the copied keychain item. Use the `SecAccessCreate` (page 105) function to create an access object or the `SecKeychainItemCopyAccess` (page 131) function to copy an access object from another keychain item. If you pass `NULL` for this parameter, the access defaults to the application creating the item.

itemCopy

On return, a pointer to a copy of the keychain item referenced by the `itemRef` parameter. You must release this object by calling the `CFRelease` function.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecKeychainItem.h

SecKeychainItemCreateFromContent

Creates a new keychain item from the supplied parameters.

```
OSStatus SecKeychainItemCreateFromContent (
   SecItemClass itemClass,
   SecKeychainAttributeList *attrList,
   UInt32 length,
   const void *data,
   SecKeychainRef keychainRef,
   SecAccessRef initialAccess,
   SecKeychainItemRef *itemRef
);
```

Parameters

itemClass
A constant identifying the class of item to create. See "Keychain Item Class Constants" (page 179) for valid constants.

attrList
A pointer to the list of attributes for the item to create.

length
The length of the buffer pointed to by the `data` parameter.

data
A pointer to a buffer containing the data to store.

keychainRef
A reference to the keychain in which to add the item. Pass `NULL` to specify the default keychain.

initialAccess
An access object for this keychain item. Use the `SecAccessCreate` (page 105) function to create an access object or the `SecKeychainItemCopyAccess` (page 131) function to copy an access object from another keychain item. If you pass `NULL` for this parameter, the access defaults to the application creating the item.

itemRef
On return, a pointer to a reference to the newly created keychain item. This parameter is optional. When the item object is no longer required, call `CFRelease` to deallocate memory occupied by the item.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Each item stored in the keychain contains data (such as a certificate), which is indexed by the item's attributes. Use this function to create a keychain item from its attributes and data. To create keychain items that hold passwords, use the `SecKeychainAddInternetPassword` (page 116) or `SecKeychainAddGenericPassword` (page 114) functions.

A `SecKeychainItemRef` object for a certificate that is stored in a keychain can be safely cast to a `SecCertificateRef` for use with the Certificate, Key, and Trust API.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecKeychainItemDelete

Deletes a keychain item from the default keychain's permanent data store.

```
OSStatus SecKeychainItemDelete (
   SecKeychainItemRef itemRef
);
```

Parameters

itemRef

A keychain item object of the item to delete. Use the `CFRelease` function when you are completely finished with this item.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
If the keychain item has not previously been added to the keychain, this function does nothing and returns `noErr`.

Do not delete a keychain item and recreate it in order to modify it; instead, use the `SecKeychainItemModifyContent` (page 142) or `SecKeychainItemModifyAttributesAndData` (page 142) function to modify an existing keychain item. When you delete a keychain item, you lose any access controls and trust settings added by the user or by other applications.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecKeychainItemExport

Exports one or more certificates, keys, or identities.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainItemExport (
   CFTypeRef keychainItemOrArray,
   SecExternalFormat outputFormat,
   SecItemImportExportFlags flags,
   const SecKeyImportExportParameters *keyParams,
   CFDataRef *exportedData
);
```

Parameters

keychainItemOrArray

The keychain item or items to export. You can export only the following types of keychain items: *SecCertificateRef*, `SecKeyRef`, and `SecIdentityRef`. If you are exporting exactly one item, you can specify a `SecKeychainItemRef` object. Otherwise this parameter is a `CFArrayRef` object containing a number of items of type `SecKeychainItemRef`.

outputFormat

The format of the external representation of the item. Set this parameter to `kSecFormatUnknown` to use the default for that item type. Possible values for this parameter and default values are enumerated in "Keychain Item Import/Export Formats" (page 182).

flags

A flag indicating whether the exported item should have PEM armour. PEM armour refers to a way of expressing binary data as an ASCII string so that it can be transferred over text-only channels such as email. Set this flag to `kSecItemPemArmour` if you want PEM armouring.

keyParams

A pointer to a structure containing a set of input parameters for the function. If no key items are being exported, these parameters are optional and you can set the `keyParams` parameter to `NULL`.

exportedData

On return, points to the external representation of the keychain item or items.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

This function works only with keys, certificates, and identities. An identity is the combination of a certificate and its associated private key. Although public keys are commonly stored in certificates, they can be stored separately in the keychain as well; for example, when you call the `SecKeyCreatePair` (page 52) function to create a key pair, both the public and private keys are stored in the keychain. Use the `SecKeychainSearchCopyNext` (page 146) function to find a key or certificate. Use the `SecIdentitySearchCopyNext` (page 49) function in the Certificate, Key, and Trust API to find an identity.

Availability

Available in Mac OS X v10.4 and later.

Declared In

`SecImportExport.h`

SecKeychainItemFreeAttributesAndData

Releases the memory used by the keychain attribute list and/or the keychain data retrieved in a call to `SecKeychainItemCopyAttributesAndData`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainItemFreeAttributesAndData (
   SecKeychainAttributeList *attrList,
   void *data
);
```

Parameters

attrList

A pointer to the attribute list to release. Pass `NULL` if there is no attribute list to release.

data

A pointer to the data buffer to release. Pass `NULL` if there is no data to release.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecKeychainItemFreeContent

Releases the memory used by the keychain attribute list and/or the keychain data retrieved in a call to the `SecKeychainItemCopyContent` (page 132) function.

```
OSStatus SecKeychainItemFreeContent (
   SecKeychainAttributeList *attrList,
   void *data
);
```

Parameters

attrList

A pointer to the attribute list to release. Pass `NULL` if there is no attribute list to release.

data

A pointer to the data buffer to release. Pass `NULL` if there is no data to release.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Because the `SecKeychainFindInternetPassword` (page 124) and `SecKeychainFindGenericPassword` (page 123) functions call the `SecKeychainItemCopyContent` (page 132) function, you must call `SecKeychainItemFreeContent` to release the data buffers after calls to those functions as well.

Because the `SecKeychainItemCopyContent` function does not allocate buffers until they are needed, you should not call the `SecKeychainItemFreeContent` function unless data is actually returned to you.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
ImageClient

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecKeychainItem.h

SecKeychainItemGetDLDBHandle

Returns the CSSM database handle for a given keychain item object.

```
OSStatus SecKeychainItemGetDLDBHandle (
   SecKeychainItemRef keyItemRef,
   CSSM_DL_DB_HANDLE *dldbHandle
);
```

Parameters

keyItemRef

A keychain item object.

dldbHandle

On return, a pointer to a CSSM database handle for the keychain database containing the given item. The handle is valid until the keychain item object is released.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychainItem.h

SecKeychainItemGetTypeID

Returns the unique identifier of the opaque type to which a SecKeychainItemRef object belongs.

```
CFTypeID SecKeychainItemGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a SecKeychainItemRef (page 160) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a SecKeychainItemRef (page 160) object. You can compare this value to the CFTypeID identifier obtained by calling the CFGetTypeID function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychainItem.h

SecKeychainItemGetUniqueRecordID

Returns a CSSM unique record for the given keychain item object.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainItemGetUniqueRecordID (
   SecKeychainItemRef itemRef,
   const CSSM_DB_UNIQUE_RECORD **uniqueRecordID
);
```

Parameters

itemRef

A keychain item object.

uniqueRecordID

On return, a pointer to a CSSM unique record for the given item. The unique record is valid until the item object is released.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychainItem.h`

SecKeychainItemImport

Imports one or more certificates, keys, or identities and adds them to a keychain.

```
OSStatus SecKeychainItemImport (
   CFDataRef importedData,
   CFStringRef fileNameOrExtension,
   SecExternalFormat *inputFormat,
   SecExternalItemType *itemType,
   SecItemImportExportFlags flags,
   const SecKeyImportExportParameters *keyParams,
   SecKeychainRef importKeychain,
   CFArrayRef *outItems
);
```

Parameters

importedData

The external representation of the items to import.

fileNameOrExtension

The name or extension of the file from which the external representation was obtained. Pass `NULL` if you don't know the name or extension.

inputFormat

On input, points to the format of the external representation. Pass `kSecFormatUnknown` if you do not know the exact format. On output, points to the format that the function has determined the external representation to be in. Pass `NULL` if you don't know the format and don't want the format returned to you.

itemType

On input, points to the item type of the item or items contained in the external representation. Pass `kSecItemTypeUnknown` if you do not know the item type. On output, points to the item type that the function has determined the external representation to contain. Pass `NULL` if you don't know the item type and don't want the type returned to you.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

flags

Unused; pass in 0.

keyParams

A pointer to a structure containing a set of input parameters for the function. If no key items are being imported, these parameters are optional and you can set the keyParams parameter to NULL.

importKeychain

A keychain object indicating the keychain to which the key or certificate should be imported. If you pass NULL, the item is not imported. Use the SecKeychainCopyDefault (page 119) function to get a reference to the default keychain. If this parameter is NULL, the kSecKeyImportOnlyOne bit in the flags parameter is ignored. Otherwise, if the kSecKeyImportOnlyOne bit is set and there is more than one key in the incoming external representation, no items are imported to the specified keychain and the error errSecMultiplePrivKeys is returned.

outItems

On output, points to an array of SecKeychainItemRef objects for the imported items. You must provide a valid pointer to a CFArrayRef object to receive this information. If you pass NULL for this parameter, the function does not return the imported items. Release this object by calling the CFRelease function when you no longer need it.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

When you pass this function a CFDataRef object containing the external representation of one or more keys, certificates, or identities, SecKeychainItemImport attempts to determine the format and contents of the data. To ensure that this process is successful, you should specify values for one or more of the parameters fileNameOrExtension, inputFormat, and itemType. To have the function add the imported items to a keychain, specify a non-NULL value for the importKeychain parameter. To have the function return SecKeychainItemRef objects for the imported items, specify a non-NULL value for the outItems parameter.

Because the SecKeychainItemImport function determines whether the item is PEM armoured by inspecting the data, the flags parameter is not used in this function call.

After the function returns, you can determine the nature of the keychain items from the values returned in the inputFormat and itemType parameters. Depending on the nature of each item, once it is imported to a keychain you can safely cast the SecKeychainItemRef object to a SecKeyRef, SecCertificateRef, or SecIdentityRef object.

Note that when you import data in PKCS12 format, typically one SecIdentityRef object is returned in the outItems parameter. The data might also include one or more SecCertificateRef objects. The output data will not include any SecKeyRef objects unless the incoming data includes a key with no matching certificate.

When the output item type is kSecItemTypeAggregate, you can use the CFGetTypeID function to determine the Core Foundation type of each item and the functions in "Getting Information About Keychain Services and Types" (page 99) to determine the keychain item type of each item. For example, the following code determines whether the item is a certificate:

```
CFTypeID theID = CFGetTypeID(theItem);
if (SecCertificateGetTypeID() == theID)
```

You can pass in NULL for both outItems and importKeychain to determine what is inside a given external data representation. When you do, the function returns the input format and the item type without modifying the data in any way.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.4 and later.

Declared In
SecImportExport.h

SecKeychainItemModifyAttributesAndData

Updates an existing keychain item after changing its attributes or data.

```
OSStatus SecKeychainItemModifyAttributesAndData (
   SecKeychainItemRef itemRef,
   const SecKeychainAttributeList *attrList,
   UInt32 length,
   const void *data
);
```

Parameters

itemRef
A reference to the keychain item to modify.

attrList
A pointer to the list of attributes to set. Pass NULL if you have no attributes to set.

length
The length of the buffer pointed to by the *data* parameter. Pass 0 if you pass NULL in the data parameter.

data
A pointer to a buffer containing the data to store. Pass NULL if you do not need to modify the data.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
The keychain item is written to the keychain's permanent data store. If the keychain item has not previously been added to a keychain, a call to this function does nothing and returns noErr.

You should pair the SecKeychainItemCopyAttributesAndData (page 131) function with the SecKeychainItemModifyAttributesAndData function, as these functions handle more attributes than are support by the old Keychain Manager and passing them into older calls yields an invalid attribute error. Use the functions SecKeychainItemModifyContent (page 142) and SecKeychainItemCopyContent (page 132) when dealing with older Keychain Manager functions.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychainItem.h

SecKeychainItemModifyContent

Updates an existing keychain item after changing its attributes and/or data.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainItemModifyContent (
   SecKeychainItemRef itemRef,
   const SecKeychainAttributeList *attrList,
   UInt32 length,
   const void *data
);
```

Parameters

itemRef

A reference to the keychain item to modify.

attrList

A pointer to the list of attributes to set. Pass `NULL` if you have no attributes to set.

length

The length of the buffer pointed to by the `data` parameter. Pass `0` if you pass `NULL` in the `data` parameter.

data

A pointer to a buffer containing the data to store. Pass `NULL` if you do not need to modify the data.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

The keychain item is written to the keychain's permanent data store. If the keychain item has not previously been added to a keychain, a call to this function does nothing and returns `noErr`.

You should pair the `SecKeychainItemModifyContent` function with the `SecKeychainItemCopyContent` (page 132) function when dealing with older Keychain Manager functions. The `SecKeychainItemCopyAttributesAndData` (page 131) and `SecKeychainItemModifyAttributesAndData` (page 142) functions handle more attributes than are support by the old Keychain Manager; however, passing them into older calls yields an invalid attribute error.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

ImageClient

Declared In

`SecKeychainItem.h`

SecKeychainItemSetAccess

Sets the access of a given keychain item.

```
OSStatus SecKeychainItemSetAccess (
   SecKeychainItemRef itemRef,
   SecAccessRef access
);
```

Parameters

itemRef

A reference to a keychain item.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

access

An access object to replace the keychain item's current access object. Use the `SecAccessCreate` (page 105) function to create an access object.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Each protected keychain item (such as a password or private key) has an associated access object. The access object contains access control list (ACL) entries, which specify trusted applications and the operations for which those operations are trusted. When an application attempts to access a keychain item for a particular purpose (such as to sign a document), the system determines whether that application is trusted to access the item for that purpose. If it is trusted, then the application is given access and no user confirmation is required. If the application is not trusted, but there is an ACL entry for that operation, then the user is asked to confirm whether the application should be given access. If there is no ACL entry for that operation, then access is denied and it is up to the calling application to try something else or to notify the user.

For more information about ACL entries, see the `SecACLCreateFromSimpleContents` (page 109) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecKeychainLock

Locks a keychain.

```
OSStatus SecKeychainLock (
   SecKeychainRef keychain
);
```

Parameters

keychain

A reference to the keychain to lock. Pass `NULL` to lock the default keychain.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).The result code `errSecNoSuchKeychain` indicates that specified keychain could not be found. The result code `errSecInvalidKeychain` indicates that the specified keychain is invalid.

Discussion
Your application should not call this function unless you are responding to a user's request to lock a keychain. In general, you should leave the keychain unlocked so that the user does not have to unlock it again in another application.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecKeychainLockAll

Locks all keychains belonging to the current user.

```
OSStatus SecKeychainLockAll (
   void
);
```

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Your application should not call this function unless you are responding to a user's request to lock a keychain. In general, you should leave the keychain unlocked so that the user does not have to unlock it again in another application.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainOpen

Opens a keychain.

```
OSStatus SecKeychainOpen (
   const char *pathName,
   SecKeychainRef *keychain
);
```

Parameters

pathName
: A constant character string representing the POSIX path to the keychain to open.

keychain
: On return, a pointer to the keychain object. The memory that this pointer occupies must be released by calling the `CFRelease` function when finished with it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
You may use this function to retrieve a pointer to a keychain object given the path of the keychain. You do not need to close the keychain, but you should release the memory that the pointer occupies when you are finished with it.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecKeychain.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecKeychainRemoveCallback

Unregisters your keychain event callback function.

```
OSStatus SecKeychainRemoveCallback (
   SecKeychainCallback callbackFunction
);
```

Parameters

callbackFunction

The callback function pointer to remove.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

Once removed, keychain events are not sent to the owner of the callback.

Availability

Available in Mac OS X v10.2 and later.

Declared In

SecKeychain.h

SecKeychainSearchCopyNext

Finds the next keychain item matching the given search criteria.

```
OSStatus SecKeychainSearchCopyNext (
   SecKeychainSearchRef searchRef,
   SecKeychainItemRef *itemRef
);
```

Parameters

searchRef

A reference to the current search criteria. The search object is created in the SecKeychainSearchCreateFromAttributes (page 147) function and must be released by calling the CFRelease function when you are done with it.

itemRef

On return, a pointer to a keychain item object of the next matching keychain item, if any. You must release this object by calling the CFRelease function.

Return Value

A result code. When there are no more items that match, errSecItemNotFound is returned. See "Keychain Services Result Codes" (page 191).

Discussion

Each item stored in the keychain contains data (such as a certificate), which is indexed by the item's attributes. Use the SecKeychainSearchCreateFromAttributes (page 147) function to specify attributes to search for. If the SecKeychainSearchCopyNext function finds a match, you can use the SecKeychainItemCopyAttributesAndData (page 131) function to retrieve the item's data.

A SecKeychainItemRef object for a certificate that is stored in a keychain can be safely cast to a SecCertificateRef for use with the Certificate, Key, and Trust API.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

To find and obtain data from a password keychain item, use the `SecKeychainFindInternetPassword` (page 124) or `SecKeychainFindGenericPassword` (page 123) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainSearch.h`

SecKeychainSearchCreateFromAttributes

Creates a search object matching a list of zero or more attributes.

```
OSStatus SecKeychainSearchCreateFromAttributes (
   CFTypeRef keychainOrArray,
   SecItemClass itemClass,
   const SecKeychainAttributeList *attrList,
   SecKeychainSearchRef *searchRef
);
```

Parameters

keychainOrArray
A reference to an array of keychains to search, a single keychain, or `NULL` to search the user's current keychain search list. Use the function `SecKeychainCopySearchList` (page 120) to retrieve the user's default search list.

itemClass
The keychain item class. See "Keychain Item Class Constants" (page 179) for valid constants.

attrList
A pointer to a list of zero or more keychain attribute records to match. Pass `NULL` to match any keychain attribute.

searchRef
On return, a pointer to the current search object. You are responsible for calling the `CFRelease` function to release this object when finished with it.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Each item stored in the keychain contains data (such as a certificate), which is indexed by the item's attributes. You look up an item in a keychain by its attributes. If you find a match, you can then retrieve the item's data. Use the search object created by this function as input to the `SecKeychainSearchCopyNext` (page 146) function to find a a keychain item and the `SecKeychainItemCopyAttributesAndData` (page 131) function to retrieve the item's data.

To find and obtain data from a password keychain item, use the `SecKeychainFindInternetPassword` (page 124) or `SecKeychainFindGenericPassword` (page 123) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainSearch.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecKeychainSearchGetTypeID

Returns the unique identifier of the opaque type to which a `SecKeychainSearchRef` object belongs.

```
CFTypeID SecKeychainSearchGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecKeychainSearchRef` (page 161) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecKeychainSearchRef` (page 161) object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainSearch.h`

SecKeychainSetAccess

Sets the application access for a keychain.

```
OSStatus SecKeychainSetAccess (
   SecKeychainRef keychain,
   SecAccessRef access
);
```

Parameters

keychain
A reference to the keychain for which to set the access. Pass `NULL` to specify the default keychain.

access
An access object of type `SecAccessRef` containing access control lists for the keychain. See "Creating an Access Object" (page 102) for instructions on creating an access object.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
In addition to the ACLs for individual keychain items, the keychain itself has ACLs. However, they are currently unused and this function is unimplemented.

Special Considerations

Although this function is available in Mac OS X v10.2 and later, it is unimplemented and returns an `unimpErr` error code if called.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecKeychainSetDefault

Sets the default keychain.

```
OSStatus SecKeychainSetDefault (
   SecKeychainRef keychain
);
```

Parameters

keychain

A reference to the keychain you wish to make the default.

Return Value

A result code. See "Keychain Services Result Codes" (page 191). The result code `errSecNoSuchKeychain` indicates that the specified keychain could not be found. The result code `errSecInvalidKeychain` indicates that the specified keychain is invalid.

Discussion

In most cases, your application should not need to set the default keychain, because this is a choice normally made by the user. You may call this function to change where a password or other keychain items are added, but since this is a user choice, you should set the default keychain back to the user specified keychain when you are done.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychain.h`

SecKeychainSetDomainDefault

Sets the default keychain for a specified preference domain.

```
OSStatus SecKeychainSetDomainDefault (
   SecPreferencesDomain domain,
   SecKeychainRef keychain
);
```

Parameters

domain

The preference domain for which you wish to set the default keychain. See "Keychain Preference Domain Constants" (page 185) for possible domain values.

keychain

A reference to the keychain you wish to set as default in the specified preference domain.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. Use this function if you want to set the default keychain for a specific preference domain. Use the `SecKeychainSetDefault` (page 149) function if you want to set the default keychain for the current preference domain. See the `SecKeychainSetPreferenceDomain` (page 150) function for a discussion of current and default preference domains.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SecKeychain.h

SecKeychainSetDomainSearchList

Sets the keychain search list for a specified preference domain.

```
OSStatus SecKeychainSetDomainSearchList (
   SecPreferencesDomain domain,
   CFArrayRef searchList
);
```

Parameters

domain

The preference domain for which you wish to set the default keychain search list. See "Keychain Preference Domain Constants" (page 185)for possible domain values.

searchList

A pointer to a keychain search list to set in the preference domain.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. Use this function if you want to set the keychain search list for a specific preference domain. Use the SecKeychainSetSearchList (page 151) function if you want to set the keychain search list for the current preference domain. See the SecKeychainSetPreferenceDomain (page 150) function for a discussion of current and default preference domains.

Availability
Available in Mac OS X v10.3 and later.

Declared In
SecKeychain.h

SecKeychainSetPreferenceDomain

Sets the keychain preference domain.

```
OSStatus SecKeychainSetPreferenceDomain (
   SecPreferencesDomain domain
);
```

Parameters

domain

The keychain preference domain to set. See "Keychain Preference Domain Constants" (page 185) for possible domain values.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. The default preference domain for system daemons (that is, for daemons running in the root session) is the system domain. The default preference domain for all other programs is the user domain.

This function changes the preference domain for all subsequent function calls; for example, if you change from the system domain to the user domain and then call `SecKeychainLock` (page 144) specifying `NULL` for the keychain, the function locks the default system keychain rather than the default user keychain. You might want to use this function, for example, when launching a system daemon from a user session so that the daemon uses system preferences rather than user preferences.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecKeychain.h`

SecKeychainSetSearchList

Specifies the list of keychains to use in the default keychain search list.

```
OSStatus SecKeychainSetSearchList (
   CFArrayRef searchList
);
```

Parameters

searchList
An array of keychain references (of type `SecKeychainRef`) specifying the list of keychains to use in the default keychain search list. Passing an empty array clears the search list.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
The default keychain search list is used by several functions; see for example SecKeychainSearchCreateFromAttributes (page 147), SecKeychainFindInternetPassword (page 124), or SecKeychainFindGenericPassword (page 123). To obtain the current default keychain search list, use the `SecKeychainCopySearchList` (page 120) function.

The default keychain search list is displayed as the keychain list in the Keychain Access utility. If you use `SecKeychainSetSearchList` to change the keychain search list, the list displayed in Keychain Access changes accordingly.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainSetSettings

Changes the settings of a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SecKeychainSetSettings (
   SecKeychainRef keychain,
   const SecKeychainSettings *newSettings
);
```

Parameters

keychain

A reference to a keychain whose settings you wish to change. Pass `NULL` to change the settings of the default keychain.

newSettings

A pointer to a keychain settings structure that defines whether the keychain locks when sleeping, or locks after a set time period of inactivity.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

SecKeychainSetUserInteractionAllowed

Enables or disables the user interface for Keychain Services functions that automatically display a user interface.

```
OSStatus SecKeychainSetUserInteractionAllowed (
   Boolean state
);
```

Parameters

state

A flag that indicates whether the Keychain Services will display a user interface. If you pass `TRUE`, user interaction is allowed. This is the default value. If `FALSE`, Keychain Services functions that normally display a user interface will instead return an error.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
Certain Keychain Services functions that require the presence of a keychain automatically display a Keychain Not Found dialog if there is none. Functions that require the keychain to be unlocked automatically display the Unlock Keychain dialog. The `SecKeychainSetUserInteractionAllowed` function enables you to control whether these functions display a user interface. By default, user interaction is permitted.

If you are writing an application that must run unattended on a server, you may wish to disable the user interface so that any subsequent keychain calls that normally bring up the unlock UI will instead return immediately with an `errSecInteractionRequired` result). In this case you must programmatically create a keychain or unlock the keychain when necessary.

Special Considerations

If you disable user interaction before calling a Keychain Services function, be sure to reenable it when you are finished. Failure to reenable user interaction will affect other clients of the Keychain Services.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

SecKeychainUnlock

Unlocks a keychain.

```
OSStatus SecKeychainUnlock (
   SecKeychainRef keychain,
   UInt32 passwordLength,
   const void *password,
   Boolean usePassword
);
```

Parameters

keychain

A reference to the keychain to unlock. Pass NULL to specify the default keychain. If you pass a locked keychain, this function displays the Unlock Keychain dialog box if you have not provided a password. If the specified keychain is currently unlocked, the Unlock Keychain dialog box is not displayed and this function returns noErr. The memory that the keychain object occupies must be released by calling the function CFRelease when you are finished with it.

passwordLength

An unsigned 32-bit integer representing the length of the password buffer.

password

A buffer containing the password for the keychain. Pass NULL if the user password is unknown. In this case, this function displays the Unlock Keychain dialog to request the user for the keychain password.

usePassword

A Boolean value indicating whether the password parameter is used. You should pass TRUE if you are passing a password or FALSE if it is to be ignored.

Return Value
A result code. See "Keychain Services Result Codes" (page 191). The result code userCanceledErr indicates that the user pressed the Cancel button in the Unlock Keychain dialog box. The result code errSecAuthFailed indicates that authentication failed because of too many unsuccessful retries. The result code errSecInteractionRequired indicates that user interaction is required to unlock the keychain.

Discussion
In most cases, your application does not need to call this function directly, since most Keychain Services functions that require an unlocked keychain do so for you. If your application needs to verify that a keychain is unlocked, call the function SecKeychainGetStatus (page 129).

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecKeychain.h

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SecTrustedApplicationCopyData

Retrieves the data of a trusted application object.

```
OSStatus SecTrustedApplicationCopyData (
   SecTrustedApplicationRef appRef,
   CFDataRef *data
);
```

Parameters

appRef

A trusted application object from which to retrieve data. Use the `SecTrustedApplicationCreateFromPath` (page 154) function to create a trusted application object.

data

On return, points to a data object for the data of the trusted application object. Call the `CFRelease` function to release this object when you are finished with it.

Return Value

A result code. See "Keychain Services Result Codes" (page 191).

Discussion

The trusted application object created by the `SecTrustedApplicationCreateFromPath` (page 154) function includes data that uniquely identifies the application, such as a cryptographic hash of the application. The operating system can use this data to verify that the application has not been altered since the trusted application object was created. When an application requests access to an item in the keychain for which it is designated as a trusted application, for example, the operating system checks this data before granting access. You can use the `SecTrustedApplicationCopyData` function to extract this data from the trusted application object for storage or for transmittal to another location (such as over a network). Use the `SecTrustedApplicationSetData` (page 155) function to insert the data back into a trusted application object. Note that this data is in a private format; there is no supported way to read or interpret it.

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecTrustedApplication.h`

SecTrustedApplicationCreateFromPath

Creates a trusted application object based on the application specified by path.

```
OSStatus SecTrustedApplicationCreateFromPath (
   const char *path,
   SecTrustedApplicationRef *app
);
```

Parameters

path

The path to the application or tool to trust. For application bundles, use the path to the bundle directory. Pass `NULL` to refer to the application or tool making this call.

app

On return, points to the newly created trusted application object. Call the `CFRelease` function to release this object when you are finished with it.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
This function creates a trusted application object, which both identifies an application and provides data that can be used to ensure that the application has not been altered since the object was created. The application object is used as input to the `SecAccessCreate` (page 105) function, which creates an access object. The access object, in turn, is used as input to the `SecKeychainItemSetAccess` (page 143) function to specify the set of applications that are trusted to access a specific keychain item.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrustedApplication.h`

SecTrustedApplicationGetTypeID

Returns the unique identifier of the opaque type to which a `SecTrustedApplication` object belongs.

```
CFTypeID SecTrustedApplicationGetTypeID (
   void
);
```

Return Value
A value that identifies the opaque type of a `SecTrustedApplicationRef` object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecTrustedApplicationRef` object. You can compare this value to the `CFTypeID` identifier obtained by calling the `CFGetTypeID` function on a specific object. These values might change from release to release or platform to platform.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrustedApplication.h`

SecTrustedApplicationSetData

Sets the data of a given trusted application object.

```
OSStatus SecTrustedApplicationSetData (
   SecTrustedApplicationRef appRef,
   CFDataRef data
);
```

Parameters

appRef
A trusted application object.

data
A reference to the data to set in the trusted application.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
If you used the `SecTrustedApplicationCopyData` (page 154) function to extract the data from a trusted application object for storage or to transmit it to a different location, you can use the `SecTrustedApplicationSetData` function to insert the data into a new trusted application object. Doing so would create an object that identifies the same application as the original trusted application object.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecTrustedApplication.h`

Callbacks

SecKeychainCallback

Defines a pointer to a customized callback function that Keychain Services calls when a keychain event has occurred.

```
typedef OSStatus (*SecKeychainCallback) (
    SecKeychainEvent keychainEvent,
    SecKeychainCallbackInfo *info,
    void *context
);
```

You would declare your keychain callback function like this if you were to name it `MyKeychainCallback`:

```
OSStatus MyKeychainCallback (
    SecKeychainEvent keychainEvent,
    SecKeychainCallbackInfo *info,
    void *context
);
```

Parameters

keychainEvent
The keychain event of which your application wishes to be notified. The type of event that can trigger your callback depends on the bit mask you passed in the `eventMask` parameter of the function `SecKeychainAddCallback` (page 114).

info
A pointer to a structure of type `SecKeychainCallbackInfo`. On return, the structure contains information about the keychain event that occurred. The Keychain Manager passes this information to your callback function through this parameter.

context
A pointer to application-defined storage that your application previously passed to the function `SecKeychainAddCallback` (page 114). You can use this value to perform operations such as tracking which instance of a function is operating.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Keychain Services Result Codes" (page 191).

Discussion
To add your callback function, use the `SecKeychainAddCallback` (page 114) function. To remove your callback function, use the `SecKeychainRemoveCallback` (page 146) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychain.h`

Data Types

SecAccessRef

Identifies a keychain or keychain item's access information.

```
typedef struct OpaqueSecAccessRef *SecAccessRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

SecACLRef

Represents information about an access control list entry.

```
typedef struct OpaqueSecTrustRef *SecACLRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

SecAFPServerSignature

Represents a 16-byte Apple File Protocol server signature block.

```
typedef UInt8 SecAFPServerSignature[16];
```

Discussion
This type represents a 16-byte Apple File Protocol server signature block. You can pass a value of this type in the `serverSignature` parameter of the functions `KCAddAppleSharePassword` and `KCFindAppleSharePassword` to represent an Apple File Protocol server signature. You can use a value of this type with the keychain item attribute constant `kSecSignatureItemAttr` to specify an Apple File Protocol server signature.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecKeychainItem.h`

SecKeychainAttribute

Contains keychain attributes.

```
struct SecKeychainAttribute
{
    SecKeychainAttrType tag;
    UInt32 length;
    void *data;
};
typedef struct SecKeychainAttribute SecKeychainAttribute;
typedef SecKeychainAttribute *SecKeychainAttributePtr;
```

Fields

`tag`
A 4-byte attribute tag. See "Keychain Item Attribute Constants" (page 170) for valid attribute types.

`length`
The length of the buffer pointed to by data.

`data`
A pointer to the attribute data.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SecBase.h`

SecKeychainAttributeInfo

Represents an attribute.

```
struct SecKeychainAttributeInfo
{
    UInt32 count;
    UInt32 *tag;
    UInt32 *format;
};
typedef struct SecKeychainAttributeInfo SecKeychainAttributeInfo;
```

Fields

`count`
The number of tag-format pairs in the respective arrays.

`tag`
A pointer to the first attribute tag in the array.

`format`
A pointer to the first attribute format in the array.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
Each tag and format item form a pair.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

SecKeychainAttributeList

Represents a list of keychain attributes.

```
struct SecKeychainAttributeList
{
    UInt32 count;
    SecKeychainAttribute *attr;
};
typedef struct SecKeychainAttributeList SecKeychainAttributeList;
```

Fields

`count`
An unsigned 32-bit integer that represents the number of keychain attributes in the array.

`attr`
A pointer to the first keychain attribute in the array.

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SecBase.h`

SecKeychainAttrType

Represents a keychain attribute type.

```
typedef OSType SecKeychainAttrType;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
`SecBase.h`

SecKeychainCallbackInfo

Contains information about a keychain event.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
struct SecKeychainCallbackInfo
{
    UInt32 version;
    SecKeychainItemRef item;
    SecKeychainRef keychain;
    pid_t pid;
};
typedef struct SecKeychainCallbackInfo SecKeychainCallbackInfo;
```

Fields

`version`

The version of this structure. See "Keychain Settings Version" (page 190) for valid constants.

`item`

A reference to the keychain item in which the event occurred. If the event did not involve an item, this field is not valid.

`keychain`

A reference to the keychain in which the event occurred. If the event did not involve a keychain, this field is not valid.

`pid`

The ID of the process that generated this event.

Discussion

This structure contains information about the keychain event of which your application wants to be notified. Keychain Services passes a pointer to this structure in the `info` parameter of your callback function. For information on how to write a keychain event callback function, see `SecKeychainCallback` (page 156).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecKeychain.h`

SecKeychainItemRef

Contains information about a keychain item.

```
typedef struct OpaqueSecKeychainItemRef *SecKeychainItemRef;
```

Discussion

A `SecKeychainItemRef` object for a certificate that is stored in a keychain can be safely cast to a `SecCertificateRef` for use with the Certificate, Key, and Trust API.

Availability

Available in Mac OS X v10.0 and later.

Declared In

`SecBase.h`

SecKeychainRef

Contains information about a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef struct OpaqueSecKeychainRef *SecKeychainRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
SecBase.h

SecKeychainSearchRef

Contains information about a keychain search.

```
typedef struct OpaqueSecKeychainSearchRef *SecKeychainSearchRef;
```

Availability
Available in Mac OS X v10.0 and later.

Declared In
SecBase.h

SecKeychainSettings

Contains information about keychain settings.

```
struct SecKeychainSettings
{
    UInt32          version;
    Boolean         lockOnSleep;
    Boolean         useLockInterval;
    UInt32          lockInterval;
};
typedef struct SecKeychainSettings    SecKeychainSettings;
```

Fields

`version`
: An unsigned 32-bit integer representing the keychain version.

`lockOnSleep`
: A Boolean value indicating whether the keychain locks when the system sleeps.

`useLockInterval`
: A Boolean value indicating whether the keychain automatically locks after a certain period of time.

`lockInterval`
: An unsigned 32-bit integer representing the number of seconds before the keychain locks. If you set `useLockInterval` to `FALSE`, set `lockInterval` to `INT_MAX` to indicate that the keychain never locks.

Discussion
This structure contains information about a keychain's settings such as locking on sleep and the lock time interval. You can use the `SecKeychainSetSettings` (page 151) and `SecKeychainCopySettings` (page 121) functions to set and copy a keychain's settings.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecKeychain.h

SecKeyImportExportParameters

Contains input parameters for import and export functions.

```
typedef struct
{
    /* for import and export */
    uint32_t                version;
    SecKeyImportExportFlags flags;
    CFTypeRef               passphrase;
    CFStringRef             alertTitle;
    CFStringRef             alertPrompt;

    /* for import only */
    SecAccessRef            accessRef;
    CSSM_KEYUSE             keyUsage;
    CSSM_KEYATTR_FLAGS      keyAttributes;
} SecKeyImportExportParameters;
```

Fields

`version`
: The version of this structure; the current value is `SEC_KEY_IMPORT_EXPORT_PARAMS_VERSION`.

`flags`
: A set of flag bits, defined in "Keychain Item Import/Export Formats" (page 182).

`passphrase`
: A password, used for `kSecFormatPKCS12` and `kSecFormatWrapped` formats only. (A password is sometimes referred to as a passphrase to emphasize the fact that a longer string that includes non-letter characters, such as numbers, punctuation, and spaces, is more secure than a simple word.) Legal types are `CFStringRef` and `CFDataRef`. PKCS12 requires passwords to be in Unicode format; passing in a `CFStringRef` as the password is the safest way to ensure that this requirement is met (and that the result is compatible with other implementations). If a `CFDataRef` object is supplied as the password for a PKCS12 export operation, the data is assumed to be in UTF8 form and is converted as appropriate.

  When importing or exporting keys (`SecKeyRef` objects) in one of the wrapped formats (`kSecFormatWrappedOpenSSL`, `kSecFormatWrappedSSH`, or `kSecFormatWrappedPKCS8`) or in `PKCS12` format, you must either explicitly specify the `passphrase` field or set the `kSecKeySecurePassphrase` bit in the `Flags` field (to prompt the user for the password).

`alertTitle`
: Title of secure password alert panel. When importing or exporting a key, if you set the `kSecKeySecurePassphrase` flag bit, you can optionally use this field to specify a string for the password panel's title bar.

`alertPrompt`
: Prompt in secure password alert panel. When importing or exporting a key, if you set the `kSecKeySecurePassphrase` flag bit, you can optionally use this field to specify a string for the prompt that appears in the password panel.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`accessRef`

Specifies the initial access controls of imported private keys. If more than one private key is being imported, all private keys get the same initial access controls. If this field is `NULL` when private keys are being imported, then the access object for the keychain item for an imported private key depends on the `kSecKeyNoAccessControl` bit in the `flags` parameter. If this bit is `0` (or `keyParams` is `NULL`), the default access control is used. If this bit is `1`, no access object is attached to the keychain item for imported private keys.

`keyUsage`

A word of bits constituting the low-level use flags for imported keys as defined in `cssmtype.h`. If this field is `0` or `keyParams` is `NULL`, the default value is `CSSM_KEYUSE_ANY`.

`keyAttributes`

A word of bits constituting the low-level attribute flags for imported keys. The default value is `CSSM_KEYATTR_SENSITIVE | CSSM_KEYATTR_EXTRACTABLE`; the `CSSM_KEYATTR_PERMANENT` bit is also added to the default if a non-`NULL` value is specified for the `importKeychain` parameter.

The following are valid values for these flags: `CSSM_KEYATTR_PERMANENT`, `CSSM_KEYATTR_SENSITIVE`, and `CSSM_KEYATTR_EXTRACTABLE`.

If the `CSSM_KEYATTR_PERMANENT` bit is set, the `importKeychain` parameter is not valid, and if any keys are found in the external representation, then the error `errSecInvalidKeychain` is returned.

The `CSSM_KEYATTR_SENSITIVE` bit indicates that the key can only be extracted in wrapped form.

Important: If you do not set the `CSSM_KEYATTR_EXTRACTABLE` bit, you cannot extract the imported key from the keychain in any form, including in wrapped form.

The `CSSM_KEYATTR_FLAGS` enumeration is defined in `cssmtype.h`. Note that the `CSSM_KEYATTR_RETURN_xxx` bits are always forced to `CSSM_KEYATTR_RETURN_REF` regardless of how they are specified in the `keyAttributes` field.

Discussion
This structure is passed in the `keyParams` parameter as input to the functions `SecKeychainItemExport` (page 136) and `SecKeychainItemImport` (page 140).

PKCS12 is an abbreviation for Public-Key Cryptography Standard # 12. This standard, by RSA Security, provides a format for external representation of keys and certificates and is described in *PKCS 12 v1.0: Personal Information Exchange Syntax*.

Availability
Available in Mac OS X v10.4 and later.

Declared In
`SecImportExport.h`

SecTrustedApplicationRef

Contains information about a trusted application.

```
typedef struct OpaqueSecTrustedApplicationRef *SecTrustedApplicationRef;
```

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecBase.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Constants

Mac OS X Keychain Services API Constants

Authorization Tag Type Constants

Defines constants that specify which operations an access control list entry applies to.

```
typedef sint32 CSSM_ACL_AUTHORIZATION_TAG;
enum {
    CSSM_ACL_AUTHORIZATION_TAG_VENDOR_DEFINED_START =
                                        0x00010000,
    CSSM_ACL_AUTHORIZATION_ANY =        CSSM_WORDID__STAR_,
    CSSM_ACL_AUTHORIZATION_LOGIN =      CSSM_WORDID_LOGIN,
    CSSM_ACL_AUTHORIZATION_GENKEY =     CSSM_WORDID_GENKEY,
    CSSM_ACL_AUTHORIZATION_DELETE =     CSSM_WORDID_DELETE,
    CSSM_ACL_AUTHORIZATION_EXPORT_WRAPPED =
                                        CSSM_WORDID_EXPORT_WRAPPED,
    CSSM_ACL_AUTHORIZATION_EXPORT_CLEAR =CSSM_WORDID_EXPORT_CLEAR,
    CSSM_ACL_AUTHORIZATION_IMPORT_WRAPPED =
                                        CSSM_WORDID_IMPORT_WRAPPED,
    CSSM_ACL_AUTHORIZATION_IMPORT_CLEAR =CSSM_WORDID_IMPORT_CLEAR,
    CSSM_ACL_AUTHORIZATION_SIGN =       CSSM_WORDID_SIGN,
    CSSM_ACL_AUTHORIZATION_ENCRYPT =    CSSM_WORDID_ENCRYPT,
    CSSM_ACL_AUTHORIZATION_DECRYPT =    CSSM_WORDID_DECRYPT,
    CSSM_ACL_AUTHORIZATION_MAC =        CSSM_WORDID_MAC,
    CSSM_ACL_AUTHORIZATION_DERIVE =     CSSM_WORDID_DERIVE
};
/* Apple-defined ACL authorization tags */
enum {
    CSSM_ACL_AUTHORIZATION_CHANGE_ACL =
                CSSM_ACL_AUTHORIZATION_TAG_VENDOR_DEFINED_START,
    CSSM_ACL_AUTHORIZATION_CHANGE_OWNER
};
```

Constants

`CSSM_ACL_AUTHORIZATION_TAG_VENDOR_DEFINED_START`

All vendor specific constants must be in the number range starting at this value.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_ANY`

No restrictions. This ACL entry applies to all operations available to the caller.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_LOGIN`

Use for a CSP (smart card) login.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`CSSM_ACL_AUTHORIZATION_GENKEY`
: Generate a key.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_DELETE`
: Delete this item.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_EXPORT_WRAPPED`
: Export a wrapped (that is, encrypted) key. This tag is checked on the key being exported; in addition, the `CSSM_ACL_AUTHORIZATION_ENCRYPT` tag is checked for any key used in the wrapping operation.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_EXPORT_CLEAR`
: Export an unencrypted key.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_IMPORT_WRAPPED`
: Import an encrypted key. This tag is checked on the key being imported; in addition, the `CSSM_ACL_AUTHORIZATION_DECRYPT` tag is checked for any key used in the unwrapping operation.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_IMPORT_CLEAR`
: Import an unencrypted key.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_SIGN`
: Digitally sign data.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_ENCRYPT`
: Encrypt data.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_DECRYPT`
: Decrypt data.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_MAC`
: Create or verify a message authentication code.

  Available in Mac OS X v10.0 and later.

  Declared in `cssmtype.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`CSSM_ACL_AUTHORIZATION_DERIVE`
Derive a new key from another key.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

`CSSM_ACL_AUTHORIZATION_CHANGE_ACL`
Change an access control list entry.

Available in Mac OS X v10.0 and later.

Declared in `cssmapple.h`.

`CSSM_ACL_AUTHORIZATION_CHANGE_OWNER`
For internal system use only. Use the `CSSM_ACL_AUTHORIZATION_CHANGE_ACL` tag for changes to owner ACL entries.

Available in Mac OS X v10.0 and later.

Declared in `cssmapple.h`.

Import/Export Parameters Version

Defines the version of an import/export parameters structure.

```
#define SEC_KEY_IMPORT_EXPORT_PARAMS_VERSION        0
```

Constants

`SEC_KEY_IMPORT_EXPORT_PARAMS_VERSION`
Defines the version number for a `SecImportExportParameters` structure used as input to the functions `SecKeychainItemExport` (page 136) and `SecKeychainItemImport` (page 140).

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

Keychain Authentication Type Constants

Defines constants you can use to identify the type of authentication to use for an Internet password.

```
typedef FourCharCode SecAuthenticationType;
enum
{
    kSecAuthenticationTypeNTLM              = AUTH_TYPE_FIX_ ('ntlm'),
    kSecAuthenticationTypeMSN               = AUTH_TYPE_FIX_ ('msna'),
    kSecAuthenticationTypeDPA               = AUTH_TYPE_FIX_ ('dpaa'),
    kSecAuthenticationTypeRPA               = AUTH_TYPE_FIX_ ('rpaa'),
    kSecAuthenticationTypeHTTPBasic         = AUTH_TYPE_FIX_ ('http'),
    kSecAuthenticationTypeHTTPDigest        = AUTH_TYPE_FIX_ ('httd'),
    kSecAuthenticationTypeHTMLForm          = AUTH_TYPE_FIX_ ('form'),
    kSecAuthenticationTypeDefault           = AUTH_TYPE_FIX_ ('dflt')
};
```

Constants

`kSecAuthenticationTypeNTLM`
Specifies Windows NT LAN Manager authentication.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecAuthenticationTypeMSN`
: Specifies Microsoft Network default authentication.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecAuthenticationTypeDPA`
: Specifies Distributed Password authentication.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecAuthenticationTypeRPA`
: Specifies Remote Password authentication.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecAuthenticationTypeHTTPBasic`
: Specifies HTTP Basic authentication. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecAuthenticationTypeHTTPDigest`
: Specifies HTTP Digest Access authentication.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecAuthenticationTypeHTMLForm`
: Specifies HTML form based authentication. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecAuthenticationTypeDefault`
: Specifies the default authentication type.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

Keychain Event Constants

Defines the keychain-related event.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef UInt32 SecKeychainEvent;
enum
{
    kSecLockEvent                = 1,
    kSecUnlockEvent              = 2,
    kSecAddEvent                 = 3,
    kSecDeleteEvent              = 4,
    kSecUpdateEvent              = 5,
    kSecPasswordChangedEvent     = 6,
    kSecDefaultChangedEvent      = 9,
    kSecDataAccessEvent          = 10,
    kSecKeychainListChangedEvent = 11
};
```

Constants

`kSecLockEvent`

Indicates a keychain was locked. It is impossible to distinguish between a lock event caused by an explicit request and one caused by a keychain that locked itself because of a timeout. Therefore, the `pid` parameter in the `SecKeychainCallbackInfo` structure does not contain useful information for this event. Note that when the login session terminates, all keychains become effectively locked; however, no `kSecLockEvent` events are generated in this case.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecUnlockEvent`

Indicates a keychain was successfully unlocked. It is impossible to distinguish between an unlock event caused by an explicit request and one that occurred automatically because the keychain was needed to perform an operation. In either case, however, the `pid` parameter in the `SecKeychainCallbackInfo` structure does return the ID of the process whose actions caused the unlock event.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecAddEvent`

Indicates an item was added to a keychain.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecDeleteEvent`

Indicates an item was deleted from a keychain.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecUpdateEvent`

Indicates a keychain item was updated.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecPasswordChangedEvent`

Indicates the keychain password was changed.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecDefaultChangedEvent`
Indicates that a different keychain was specified as the default.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecDataAccessEvent`
Indicates a process has accessed a keychain item's data.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecKeychainListChangedEvent`
Indicates the list of keychains has changed.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

Keychain Event Mask Constants

Defines bit masks for keychain event constants

```
typedef UInt32 SecKeychainEventMask;
enum
{
    kSecLockEventMask            = 1 << kSecLockEvent,
    kSecUnlockEventMask          = 1 << kSecUnlockEvent,
    kSecAddEventMask             = 1 << kSecAddEvent,
    kSecDeleteEventMask          = 1 << kSecDeleteEvent,
    kSecUpdateEventMask          = 1 << kSecUpdateEvent,
    kSecPasswordChangedEventMask = 1 << kSecPasswordChangedEvent,
    kSecDefaultChangedEventMask  = 1 << kSecDefaultChangedEvent,
    kSecDataAccessEventMask      = 1 << kSecDataAccessEvent,
    kSecKeychainListChangedMask  = 1 << kSecKeychainListChangedEvent,
    kSecEveryEventMask           = 0xffffffff
};
```

Constants

`kSecLockEventMask`
If the bit specified by this mask is set, your callback function is invoked when a keychain is locked.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecUnlockEventMask`
If the bit specified by this mask is set, your callback function is invoked when a keychain is unlocked.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecAddEventMask`
If the bit specified by this mask is set, your callback function is invoked when an item is added to a keychain.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecDeleteEventMask`
: If the bit specified by this mask is set, your callback function is invoked when an item is deleted from a keychain.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

`kSecUpdateEventMask`
: If the bit specified by this mask is set, your callback function is invoked when a keychain item is updated.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

`kSecPasswordChangedEventMask`
: If the bit specified by this mask is set, your callback function is invoked when the keychain password is changed.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

`kSecDefaultChangedEventMask`
: If the bit specified by this mask is set, your callback function is invoked when a different keychain is specified as the default.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

`kSecDataAccessEventMask`
: If the bit specified by this mask is set, your callback function is invoked when a process accesses a keychain item's data.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

`kSecKeychainListChangedMask`
: If the bit specified by this mask is set, your callback function is invoked when a keychain list is changed.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

`kSecEveryEventMask`
: If all the bits are set, your callback function is invoked whenever any event occurs.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychain.h`.

Keychain Item Attribute Constants

Specifies a keychain item's attributes.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef FourCharCode SecItemAttr;
enum
{
    kSecCreationDateItemAttr        = 'cdat',
    kSecModDateItemAttr             = 'mdat',
    kSecDescriptionItemAttr         = 'desc',
    kSecCommentItemAttr             = 'icmt',
    kSecCreatorItemAttr             = 'crtr',
    kSecTypeItemAttr                = 'type',
    kSecScriptCodeItemAttr          = 'scrp',
    kSecLabelItemAttr               = 'labl',
    kSecInvisibleItemAttr           = 'invi',
    kSecNegativeItemAttr            = 'nega',
    kSecCustomIconItemAttr          = 'cusi',
    kSecAccountItemAttr             = 'acct',
    kSecServiceItemAttr             = 'svce',
    kSecGenericItemAttr             = 'gena',
    kSecSecurityDomainItemAttr      = 'sdmn',
    kSecServerItemAttr              = 'srvr',
    kSecAuthenticationTypeItemAttr  = 'atyp',
    kSecPortItemAttr                = 'port',
    kSecPathItemAttr                = 'path',
    kSecVolumeItemAttr              = 'vlme',
    kSecAddressItemAttr             = 'addr',
    kSecSignatureItemAttr           = 'ssig',
    kSecProtocolItemAttr            = 'ptcl',
    kSecCertificateType             = 'ctyp',
    kSecCertificateEncoding         = 'cenc',
    kSecCrlType                     = 'crtp',
    kSecCrlEncoding                 = 'crnc',
    kSecAlias                       = 'alis'
};
```

Constants

`kSecCreationDateItemAttr`

Identifies the creation date attribute. You use this tag to set or get a value of type `UInt32` that indicates the date the item was created.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecModDateItemAttr`

Identifies the modification date attribute. You use this tag to set or get a value of type `UInt32` that indicates the last time the item was updated.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecDescriptionItemAttr`

Identifies the description attribute. You use this tag to set or get a value of type `string` that represents a user-visible string describing this particular kind of item, for example “disk image password”.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecCommentItemAttr`

Identifies the comment attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing comments for this item.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecCreatorItemAttr`

Identifies the creator attribute. You use this tag to set or get a value that represents the item's creator.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecTypeItemAttr`

Identifies the type attribute. You use this tag to set or get a value that represents the item's type.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecScriptCodeItemAttr`

Identifies the script code attribute. You use this tag to set or get a value of type `ScriptCode` that represents the script code for all strings. Use of this attribute is deprecated; string attributes should be stored in UTF-8 encoding.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecLabelItemAttr`

Identifies the label attribute. You use this tag to set or get a value of type `string` that represents a user-editable string containing the label for this item.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecInvisibleItemAttr`

Identifies the invisible attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item is invisible.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecNegativeItemAttr`

Identifies the negative attribute. You use this tag to set or get a value of type `Boolean` that indicates whether there is a valid password associated with this keychain item. This is useful if your application doesn't want a password for some particular service to be stored in the keychain, but prefers that it always be entered by the user. The item, which is typically invisible and with zero-length data, acts as a placeholder.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecCustomIconItemAttr`
: Identifies the custom icon attribute. You use this tag to set or get a value of type `Boolean` that indicates whether the item has an application-specific icon. To do this, you must also set the attribute value identified by the tag `kSecTypeItemAttr` to a file type for which there is a corresponding icon in the desktop database, and set the attribute value identified by the tag `kSecCreatorItemAttr` to an appropriate application creator type. If a custom icon corresponding to the item's type and creator can be found in the desktop database, it will be displayed by Keychain Access. Otherwise, default icons are used.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

`kSecAccountItemAttr`
: Identifies the account attribute. You use this tag to set or get a string that represents the user account. It also applies to generic and AppleShare passwords.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

`kSecServiceItemAttr`
: Identifies the service attribute. You use this tag to set or get a string that represents the service associated with this item, for example, "iTools". This is unique to generic password attributes.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

`kSecGenericItemAttr`
: Identifies the generic attribute. You use this tag to set or get a value of untyped bytes that represents a user-defined attribute. This is unique to generic password attributes.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

`kSecSecurityDomainItemAttr`
: Identifies the security domain attribute. You use this tag to set or get a value that represents the Internet security domain. This is unique to Internet password attributes.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

`kSecServerItemAttr`
: Identifies the server attribute. You use this tag to set or get a string that represents the Internet server's domain name or IP address. This is unique to Internet password attributes.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

`kSecAuthenticationTypeItemAttr`
: Identifies the authentication type attribute. You use this tag to set or get a value of type `SecAuthenticationType` that represents the Internet authentication scheme. For possible authentication values, see "Keychain Authentication Type Constants" (page 166). This is unique to Internet password attributes.

  Available in Mac OS X v10.2 and later.

  Declared in `SecKeychainItem.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecPortItemAttr`
Identifies the port attribute. You use this tag to set or get a value of type `UInt32` that represents the Internet port number. This is unique to Internet password attributes.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecPathItemAttr`
Identifies the path attribute. You use this tag to set or get a value that represents the path. This is unique to Internet password attributes.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecVolumeItemAttr`
Identifies the volume attribute. You use this tag to set or get a value that represents the AppleShare volume. This is unique to AppleShare password attributes.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecAddressItemAttr`
Identifies the address attribute. You use this tag to set or get a value of type `string` that represents the AppleTalk zone name, or the IP or domain name that represents the server address. This is unique to AppleShare password attributes.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecSignatureItemAttr`
Identifies the server signature attribute. You use this tag to set or get a value of type `SecAFPServerSignature` (page 157) that represents the server signature block. This is unique to AppleShare password attributes.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecProtocolItemAttr`
Identifies the protocol attribute. You use this tag to set or get a value of type `SecProtocolType` that represents the Internet protocol. For possible protocol type values, see "Keychain Protocol Type Constants" (page 186). This is unique to AppleShare and Internet password attributes.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecCertificateType`
Indicates a `CSSM_CERT_TYPE` type.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecCertificateEncoding`
Indicates a `CSSM_CERT_ENCODING` type.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecCrlType`
Indicates a `CSSM_CRL_TYPE` type.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecCrlEncoding`
Indicates a `CSSM_CRL_ENCODING` type.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecAlias`
Indicates an alias.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

Discussion
Not all of these attributes are used for all types of items. Which set of attributes exist for each type of item is documented in the "Data Storage Library Services" chapter of *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm) for standard items and in the DL section of the *Security Release Notes* for Apple-defined item types (if any).

To obtain information about a certificate, use the CDSA Certificate Library (CL) API. To obtain information about a key, use the `SecKeyGetCSSMKey` function and the CDSA Cryptographic Service Provider (CSP) API.

For attributes for keys, see "Keychain Item Attribute Constants For Keys" (page 175).

Keychain Item Attribute Constants For Keys

Specifies the attributes for a key item in a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
enum
{
    kSecKeyKeyClass         =0,
    kSecKeyPrintName        =1,
    kSecKeyAlias            =2,
    kSecKeyPermanent        =3,
    kSecKeyPrivate          =4,
    kSecKeyModifiable       =5,
    kSecKeyLabel            =6,
    kSecKeyApplicationTag   =7,
    kSecKeyKeyCreator       =8,
    kSecKeyKeyType          =9,
    kSecKeyKeySizeInBits    =10,
    kSecKeyEffectiveKeySize =11,
    kSecKeyStartDate        =12,
    kSecKeyEndDate          =13,
    kSecKeySensitive        =14,
    kSecKeyAlwaysSensitive  =15,
    kSecKeyExtractable      =16,
    kSecKeyNeverExtractable =17,
    kSecKeyEncrypt          =18,
    kSecKeyDecrypt          =19,
    kSecKeyDerive           =20,
    kSecKeySign             =21,
    kSecKeyVerify           =22,
    kSecKeySignRecover      =23,
    kSecKeyVerifyRecover    =24,
    kSecKeyWrap             =25,
    kSecKeyUnwrap           =26
};
```

Constants

`kSecKeyKeyClass`

Type uint32 (`CSSM_KEYCLASS`); value is one of `CSSM_KEYCLASS_PUBLIC_KEY`, `CSSM_KEYCLASS_PRIVATE_KEY` or `CSSM_KEYCLASS_SESSION_KEY`.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyPrintName`

Type blob; human readable name of the key. Same as `kSecLabelItemAttr` for normal keychain items.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyAlias`

Type blob; currently unused.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyPermanent`

Type uint32; value is nonzero. This key is permanent (stored in some keychain) and is always `1`.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecKeyPrivate`
Type uint32; value is nonzero. This key is protected by a user login, a password, or both.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyModifiable`
Type uint32; value is nonzero. Attributes of this key can be modified.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyLabel`
Type blob; for private and public keys this contains the hash of the public key. This is used to associate certificates and keys. Its value matches the value of the `kSecPublicKeyHashItemAttr` attribute of a certificate and it's used to construct an identity from a certificate and a key. For symmetric keys this is whatever the creator of the key passed in when they generated the key.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyApplicationTag`
Type blob; currently unused.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyKeyCreator`
Type data. The data points to a `CSSM_GUID` structure representing the module ID of the CSP owning this key.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyKeyType`
Type uint32; value is a CSSM algorithm (`CSSM_ALGORITHMS`) representing the algorithm associated with this key.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyKeySizeInBits`
Type uint32; value is the number of bits in this key.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyEffectiveKeySize`
Type uint32; value is the effective number of bits in this key. For example, a DES key has a key size in bits (`kSecKeyKeySizeInBits`) of 64 but a value for `kSecKeyEffectiveKeySize` of 56.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyStartDate`
Type `CSSM_DATE`. Earliest date at which this key may be used. If the value is all zeros or not present, no restriction applies.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecKeyEndDate`
: Type `CSSM_DATE`. Latest date at which this key may be used. If the value is all zeros or not present, no restriction applies.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeySensitive`
: Type uint32; value is nonzero. This key cannot be wrapped with `CSSM_ALGID_NONE`.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyAlwaysSensitive`
: Type uint32; value is nonzero. This key has always been marked sensitive.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyExtractable`
: Type uint32; value is nonzero. This key can be wrapped.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyNeverExtractable`
: Type uint32; value is nonzero. This key was never marked extractable.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyEncrypt`
: Type uint32; value is nonzero. This key can be used in an encrypt operation.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyDecrypt`
: Type uint32; value is nonzero. This key can be used in a decrypt operation.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyDerive`
: Type uint32; value is nonzero. This key can be used in a key derivation operation.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeySign`
: Type uint32, value is nonzero. This key can be used in a sign operation.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

`kSecKeyVerify`
: Type uint32, value is nonzero. This key can be used in a verify operation.

Available in Mac OS X v10.4 and later.

Declared in `SecKey.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecKeySignRecover`
Type uint32.
Available in Mac OS X v10.4 and later.
Declared in `SecKey.h`.

`kSecKeyVerifyRecover`
Type uint32. This key can unwrap other keys.
Available in Mac OS X v10.4 and later.
Declared in `SecKey.h`.

`kSecKeyWrap`
Type uint32; value is nonzero. This key can wrap other keys.
Available in Mac OS X v10.4 and later.
Declared in `SecKey.h`.

`kSecKeyUnwrap`
Type uint32; value is nonzero. This key can unwrap other keys.
Available in Mac OS X v10.4 and later.
Declared in `SecKey.h`.

Discussion
For attributes for items other than keys, see "Keychain Item Attribute Constants" (page 170).

Keychain Item Class Constants

Specifies a keychain item's class code.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef FourCharCode SecItemClass;
enum
{
    /* SecKeychainItem.h */
    kSecInternetPasswordItemClass   = 'inet',
    kSecGenericPasswordItemClass    = 'genp',
    kSecAppleSharePasswordItemClass = 'ashp',
    kSecCertificateItemClass        =
                            CSSM_DL_DB_RECORD_X509_CERTIFICATE,
};
enum
{
    /* Record Types defined in The Open Group Application Name Space  */
    /* cssmtype.h */
    CSSM_DL_DB_RECORD_PUBLIC_KEY =
                            CSSM_DB_RECORDTYPE_OPEN_GROUP_START  + 5,
    CSSM_DL_DB_RECORD_PRIVATE_KEY =
                            CSSM_DB_RECORDTYPE_OPEN_GROUP_START  + 6,
    CSSM_DL_DB_RECORD_SYMMETRIC_KEY =
                            CSSM_DB_RECORDTYPE_OPEN_GROUP_START  + 7,
    CSSM_DL_DB_RECORD_ALL_KEYS =
                            CSSM_DB_RECORDTYPE_OPEN_GROUP_START  + 8
};
```

Constants

`kSecInternetPasswordItemClass`
Indicates that the item is an Internet password.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecGenericPasswordItemClass`
Indicates that the item is a generic password.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecAppleSharePasswordItemClass`
Indicates that the item is an AppleShare password.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`kSecCertificateItemClass`
Indicates that the item is an X509 certificate.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychainItem.h`.

`CSSM_DL_DB_RECORD_PUBLIC_KEY`
Indicates that the item is a public key of a public-private pair.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

`CSSM_DL_DB_RECORD_PRIVATE_KEY`
Indicates that the item is a private key of a public-private pair.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`CSSM_DL_DB_RECORD_SYMMETRIC_KEY`
Indicates that the item is a private key used for symmetric-key encryption.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

`CSSM_DL_DB_RECORD_ALL_KEYS`
The item can be any type of key; used for searches only.

Available in Mac OS X v10.0 and later.

Declared in `cssmtype.h`.

Discussion
These enumerations define constants your application can use to specify the type of the keychain item you wish to create, dispose, add, delete, update, copy, or locate. You can also use these constants with the tag constant `SecItemAttr`.

Declared In
`SecKeychainItem.h, cssmtype.h.`

Keychain Item Import/Export Flags

Defines values for import and export flags.

```
enum
{
    kSecItemPemArmour           = 0x00000001,
};
typedef uint32_t SecItemImportExportFlags;
```

Constants
`kSecItemPemArmour`
The exported data should have PEM armour.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

Discussion
This enumeration lists values used by the `flags` parameter of the functions `SecKeychainItemExport` (page 136) and `SecKeychainItemImport` (page 140).

PEM armour refers to a way of expressing binary data as an ASCII string so that it can be transferred over text-only channels such as email. (PEM stands for an Internet standard, Privacy Enhanced Mail.)

Keychain Item Import/Export Parameter Flags

Defines values for the `flags` field of the import/export parameters.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
enum
{
    kSecKeyImportOnlyOne        = 0x00000001,
    kSecKeySecurePassphrase     = 0x00000002,
    kSecKeyNoAccessControl      = 0x00000004
};
typedef uint32_t SecKeyImportExportFlags;
```

Constants

`kSecKeyImportOnlyOne`

Prevents the importing of more than one private key by the `SecKeychainItemImport` (page 140) function. If the `importKeychain` parameter is `NULL`, this bit is ignored. Otherwise, if this bit is set and there is more than one key in the incoming external representation, no items are imported to the specified keychain and the error `errSecMultipleKeys` is returned.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecKeySecurePassphrase`

When set, the password for import or export is obtained by user prompt. (A password is sometimes referred to as a passphrase to emphasize the fact that a longer string that includes non-letter characters, such as numbers, punctuation, and spaces, is more secure than a simple word.) Otherwise, you must provide the password in the `passphrase` field of the `SecKeyImportExportParameters` structure. A user-supplied password is preferred, because it avoids having the cleartext password appear in the application's address space at any time.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecKeyNoAccessControl`

When set, imported private keys have no access object attached to them. In the absence of both this bit and the `accessRef` field in `SecKeyImportExportParameters`, imported private keys are given default access controls.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

Discussion

These flags are used as input to the import/export parameters structure (`SecKeyImportExportParameters` (page 162), which in turn is used as input to the functions `SecKeychainItemExport` (page 136) and `SecKeychainItemImport` (page 140).

Keychain Item Import/Export Formats

Specifies the format of an item after export from or before import to the keychain.

```
enum
{
    kSecFormatUnknown = 0,

    /* Asymmetric Key Formats */
    kSecFormatOpenSSL,
    kSecFormatSSH,      //not supported
    kSecFormatBSAFE,
```


2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
    /* Symmetric Key Formats */
    kSecFormatRawKey,

    /* Formats for wrapped symmetric and private keys */
    kSecFormatWrappedPKCS8,
    kSecFormatWrappedOpenSSL,
    kSecFormatWrappedSSH,   //not supported
    kSecFormatWrappedLSH,   //not supported

    /* Formats for certificates */
    kSecFormatX509Cert,

    /* Aggregate Types */
    kSecFormatPEMSequence,
    kSecFormatPKCS7,
    kSecFormatPKCS12,
    kSecFormatNetscapeCertSequence
};
typedef uint32_t SecExternalFormat;
```

Constants

`kSecFormatUnknown`

When importing, indicates the format is unknown. When exporting, use the default format for the item. For asymmetric keys, the default is `kSecFormatOpenSSL`. For symmetric keys, the default is `kSecFormatRawKey`. For certificates, the default is `kSecFormatX509Cert`. For multiple items, the default is `kSecFormatPEMSequence`.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatOpenSSL`

Format for asymmetric (public/private) keys. OpenSSL is an open source toolkit for Secure Sockets Layer (SSL) and Transport Layer Security (TLS). Also known as X.509 for public keys.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatSSH`

Not supported.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatBSAFE`

Format for asymmetric keys. BSAFE is a standard from RSA Security for encryption, digital signatures, and privacy.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatRawKey`

Format for symmetric keys. Raw, unformatted key bits. This is the default for symmetric keys.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecFormatWrappedPKCS8`
Format for wrapped symmetric and private keys. PKCS8 is the Private-Key Information Syntax Standard from RSA Security.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatWrappedOpenSSL`
Format for wrapped symmetric and private keys. OpenSSL is an open-source toolkit for Secure Sockets Layer (SSL) and Transport Layer Security (TLS).

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatWrappedSSH`
Not supported.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatWrappedLSH`
Not supported.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatX509Cert`
Format for certificates. DER (distinguished encoding rules) encoded. X.509 is a standard for digital certificates from the International Telecommunication Union (ITU). This is the default for certificates.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatPEMSequence`
Sequence of certificates and keys with PEM armour. PEM armour refers to a way of expressing binary data as an ASCII string so that it can be transferred over text-only channels such as email. This is the default format for multiple items.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatPKCS7`
Sequence of certificates, no PEM armour. PKCS7 is the Cryptographic Message Syntax Standard from RSA Security, Inc.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatPKCS12`
Set of certificates and private keys. PKCS12 is the Personal Information Exchange Syntax from RSA Security, Inc.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

`kSecFormatNetscapeCertSequence`
Set of certificates in the Netscape Certificate Sequence format.

Available in Mac OS X v10.4 and later.

Declared in `SecImportExport.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Keychain Item Type When Importing

Specifies the type of keychain item being imported.

```
enum {
    kSecItemTypeUnknown,            /* caller doesn't know what  this is */
    kSecItemTypePrivateKey,
    kSecItemTypePublicKey,
    kSecItemTypeSessionKey,
    kSecItemTypeCertificate,
    kSecItemTypeAggregate
};
typedef uint32_t SecExternalItemType;
```

Constants

`kSecItemTypePrivateKey`
Indicates a private key.
Available in Mac OS X v10.4 and later.
Declared in `SecImportExport.h`.

`kSecItemTypePublicKey`
Indicates a public key.
Available in Mac OS X v10.4 and later.
Declared in `SecImportExport.h`.

`kSecItemTypeSessionKey`
Indicates a session key.
Available in Mac OS X v10.4 and later.
Declared in `SecImportExport.h`.

`kSecItemTypeCertificate`
Indicates a certificate.
Available in Mac OS X v10.4 and later.
Declared in `SecImportExport.h`.

`kSecItemTypeAggregate`
Indicates a set of certificates or certificates and private keys, such as `PKCS7`, `PKCS12`, or `kSecFormatPEMSequence` formats (see "Keychain Item Import/Export Formats" (page 182)).
Available in Mac OS X v10.4 and later.
Declared in `SecImportExport.h`.

Keychain Preference Domain Constants

Defines constants for the keychain preference domains.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef enum {
    kSecPreferencesDomainUser,
    kSecPreferencesDomainSystem,
    kSecPreferencesDomainCommon,
    kSecPreferencesDomainAlternate } SecPreferencesDomain;
```

Constants

`kSecPreferencesDomainUser`

Indicates the user preference domain preferences.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecPreferencesDomainSystem`

Indicates the system or daemon preference domain preferences.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecPreferencesDomainCommon`

Indicates the preferences are common to everyone.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecPreferencesDomainAlternate`

Indicates an alternate preference domain preferences.

Available in Mac OS X v10.3 through Mac OS X v10.3.

Declared in `SecKeychain.h`.

Discussion

A preference domain is a set of security-related preferences, such as the default keychain and the current keychain search list. The default preference domain for system daemons (that is, for daemons running in the root session) is the system domain. The default preference domain for all other programs is the user domain. A common preference appears for all users and the system; for example, if you add a keychain to the keychain search list using `kSecPreferencesDomainCommon` for the preference domain, the keychain is added to the search list for all users and the system.

Keychain Protocol Type Constants

Defines the protocol type associated with an AppleShare or Internet password.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef FourCharCode SecProtocolType;
enum
{
    kSecProtocolTypeFTP         = 'ftp ',
    kSecProtocolTypeFTPAccount  = 'ftpa',
    kSecProtocolTypeHTTP        = 'http',
    kSecProtocolTypeIRC         = 'irc ',
    kSecProtocolTypeNNTP        = 'nntp',
    kSecProtocolTypePOP3        = 'pop3',
    kSecProtocolTypeSMTP        = 'smtp',
    kSecProtocolTypeSOCKS       = 'sox ',
    kSecProtocolTypeIMAP        = 'imap',
    kSecProtocolTypeLDAP        = 'ldap',
    kSecProtocolTypeAppleTalk   = 'atlk',
    kSecProtocolTypeAFP         = 'afp ',
    kSecProtocolTypeTelnet      = 'teln',
    kSecProtocolTypeSSH         = 'ssh ',
    kSecProtocolTypeFTPS        = 'ftps',
    kSecProtocolTypeHTTPS       = 'htps',
    kSecProtocolTypeHTTPProxy   = 'htpx',
    kSecProtocolTypeHTTPSProx   = 'htsx',
    kSecProtocolTypeFTPProxy    = 'ftpx',
    kSecProtocolTypeSMB         = 'smb ',
    kSecProtocolTypeRTSP        = 'rtsp',
    kSecProtocolTypeRTSPProxy   = 'rtsx',
    kSecProtocolTypeDAAP        = 'daap',
    kSecProtocolTypeEPPC        = 'eppc',
    kSecProtocolTypeIPP         = 'ipp ',
    kSecProtocolTypeNNTPS       = 'ntps',
    kSecProtocolTypeLDAPS       = 'ldps',
    kSecProtocolTypeTelnetS     = 'tels',
    kSecProtocolTypeIMAPS       = 'imps',
    kSecProtocolTypeIRCS        = 'ircs',
    kSecProtocolTypePOP3S       = 'pops'
};
```

Constants

`kSecProtocolTypeFTP`

Indicates FTP.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeFTPAccount`

Indicates a client side FTP account. The usage of this constant is deprecated as of Mac OS X v10.3.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeHTTP`

Indicates HTTP.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeIRC`

Indicates IRC.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecProtocolTypeNNTP`
Indicates NNTP.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypePOP3`
Indicates POP3.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeSMTP`
Indicates SMTP.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeSOCKS`
Indicates SOCKS.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeIMAP`
Indicates IMAP.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeLDAP`
Indicates LDAP.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeAppleTalk`
Indicates AFP over AppleTalk.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeAFP`
Indicates AFP over TCP.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeTelnet`
Indicates Telnet.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

`kSecProtocolTypeSSH`
Indicates SSH.
Available in Mac OS X v10.2 and later.
Declared in `SecKeychain.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecProtocolTypeFTPS`
Indicates FTP over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeHTTPS`
Indicates HTTP over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeHTTPProxy`
Indicates HTTP proxy. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeHTTPSProxy`
Indicates HTTPS proxy. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeFTPProxy`
Indicates FTP proxy. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeSMB`
Indicates SMB. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeRTSP`
Indicates RTSP. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeRTSPProxy`
Indicates RTSP proxy. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeDAAP`
Indicates DAAP. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeEPPC`
Indicates Remote Apple Events. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSecProtocolTypeIPP`
Indicates IPP. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeNNTPS`
Indicates NNTP over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeLDAPS`
Indicates LDAP over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeTelnetS`
Indicates Telnet over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeIMAPS`
Indicates IMAP4 over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypeIRCS`
Indicates IRC over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

`kSecProtocolTypePOP3S`
Indicates POP3 over TLS/SSL. This constant is available in Mac OS X v10.3 and later.

Available in Mac OS X v10.3 and later.

Declared in `SecKeychain.h`.

Keychain Settings Version

Defines the keychain settings version.

```
#define SEC_KEYCHAIN_SETTINGS_VERS1 1
```

Constants

`SEC_KEYCHAIN_SETTINGS_VERS1`
Defines the keychain settings version.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

Keychain Status Masks

Defines the current status of a keychain.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef UInt32 SecKeychainStatus;
enum
{
    kSecUnlockStateStatus        = 1,
    kSecReadPermStatus           = 2,
    kSecWritePermStatus          = 4
};
```

Constants

`kSecUnlockStateStatus`

Indicates the keychain is unlocked.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecReadPermStatus`

Indicates the keychain is readable.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

`kSecWritePermStatus`

Indicates the keychain is writable.

Available in Mac OS X v10.2 and later.

Declared in `SecKeychain.h`.

Discussion

You can use these masks in combination. For example, a keychain may be both readable and writable.

Result Codes

The most common result codes returned by Keychain Services are listed in the table below. The assigned error space for Keychain Services is discontinuous: –25240 through –25279 and –25290 through –25329. Keychain Item Services may also return `noErr` (0) or `paramErr` (–50), or CSSM result codes (see *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm)).

Result Code	Value	Description
`errSecNotAvailable`	–25291	No trust results are available. Available in Mac OS X v10.2 and later.
`errSecReadOnly`	–25292	Read only error. Available in Mac OS X v10.2 and later.
`errSecAuthFailed`	–25293	Authorization/Authentication failed. Available in Mac OS X v10.2 and later.
`errSecNoSuchKeychain`	–25294	The keychain does not exist. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSecInvalidKeychain	–25295	The keychain is not valid. Available in Mac OS X v10.2 and later.
errSecDuplicateKeychain	–25296	A keychain with the same name already exists. Available in Mac OS X v10.2 and later.
errSecDuplicateCallback	–25297	More than one callback of the same name exists. Available in Mac OS X v10.2 and later.
errSecInvalidCallback	–25298	The callback is not valid. Available in Mac OS X v10.2 and later.
errSecDuplicateItem	–25299	The item already exists. Available in Mac OS X v10.2 and later.
errSecItemNotFound	–25300	The item cannot be found. Available in Mac OS X v10.2 and later.
errSecBufferTooSmall	–25301	The buffer is too small. Available in Mac OS X v10.2 and later.
errSecDataTooLarge	–25302	The data is too large for the particular data type. Available in Mac OS X v10.2 and later.
errSecNoSuchAttr	–25303	The attribute does not exist. Available in Mac OS X v10.2 and later.
errSecInvalidItemRef	–25304	The item reference is invalid. Available in Mac OS X v10.2 and later.
errSecInvalidSearchRef	–25305	The search reference is invalid. Available in Mac OS X v10.2 and later.
errSecNoSuchClass	–25306	The keychain item class does not exist. Available in Mac OS X v10.2 and later.
errSecNoDefaultKeychain	–25307	A default keychain does not exist. Available in Mac OS X v10.2 and later.
errSecInteractionNotAllowed	–25308	Interaction with the Security Server is not allowed. Available in Mac OS X v10.2 and later.
errSecReadOnlyAttr	–25309	The attribute is read only. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSecWrongSecVersion	–25310	The version is incorrect. Available in Mac OS X v10.2 and later.
errSecKeySizeNotAllowed	–25311	The key size is not allowed. Available in Mac OS X v10.2 and later.
errSecNoStorageModule	–25312	There is no storage module available. Available in Mac OS X v10.2 and later.
errSecNoCertificateModule	–25313	There is no certificate module available. Available in Mac OS X v10.2 and later.
errSecNoPolicyModule	–25314	There is no policy module available. Available in Mac OS X v10.2 and later.
errSecInteractionRequired	–25315	User interaction is required. Available in Mac OS X v10.2 and later.
errSecDataNotAvailable	–25316	The data is not available. Available in Mac OS X v10.2 and later.
errSecDataNotModifiable	–25317	The data is not modifiable. Available in Mac OS X v10.2 and later.
errSecCreateChainFailed	–25318	The attempt to create a certificate chain failed. Available in Mac OS X v10.2 and later.
errSecInvalidPrefsDomain	–25319	The preference domain specified is invalid. This error is available in Mac OS X v10.3 and later. Available in Mac OS X v10.3 and later.
errSecACLNotSimple	–25240	The access control list is not in standard simple form. Available in Mac OS X v10.2 and later.
errSecPolicyNotFound	–25241	The policy specified cannot be found. Available in Mac OS X v10.2 and later.
errSecInvalidTrustSetting	–25242	The trust setting is invalid. Available in Mac OS X v10.2 and later.
errSecNoAccessForItem	–25243	The specified item has no access control. Available in Mac OS X v10.2 and later.
errSecInvalidOwnerEdit	–25244	An invalid attempt to change the owner of an item. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSecTrustNotAvailable	–25245	No trust results are available. Available in Mac OS X v10.3 and later.
errSecUnsupportedFormat	–25256	The specified import or export format is not supported. Available in Mac OS X v10.4 and later.
errSecUnknownFormat	–25257	The item you are trying to import has an unknown format. Available in Mac OS X v10.4 and later.
errSecKeyIsSensitive	–25258	The key must be wrapped to be exported. Available in Mac OS X v10.4 and later.
errSecMultiplePrivKeys	–25259	An attempt was made to import multiple private keys. Available in Mac OS X v10.4 and later.
errSecPassphraseRequired	–25260	A password is required for import or export. Available in Mac OS X v10.4 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Other References

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Secure Transport Reference

Framework:	Security/Security.h
Declared in	SecureTransport.h CipherSuite.h

Overview

This document describes the public API for an implementation of the protocols Secure Sockets Layer version 3.0 and Transport Layer Security version 1.0.

There are no transport layer dependencies in this library; it can be used with BSD Sockets and Open Transport, among other protocols. To use this library, you must provide callback functions to perform the actual I/O on underlying network connections. You are also responsible for setting up raw network connections; you pass in an opaque reference to the underlying (connected) entity at the start of an SSL session in the form of an `SSLConnectionRef` (page 234) object.

The following terms are used in this document:

- A *client* is the initiator of an SSL session. The canonical example of a client is a web browser communicating with an HTTPS URL.
- A *server* is an entity that accepts requests for SSL sessions made by clients. An example is a secure web server.
- An SSL *session* is bounded by calls to the functions `SSLHandshake` (page 216) and `SSLClose` (page 202). An active session is in some state between these two calls, inclusive.
- An SSL *session context*, or `SSLContextRef` (page 234), is an opaque reference to the state associated with one session. A session context cannot be reused for multiple sessions.

Most applications need only a few of the functions in this API, which are normally called in the following sequence:

1. Preparing for a session
    a. Call `SSLNewContext` (page 217) to create a new SSL session context.
    b. Write I/O functions and call `SSLSetIOFuncs` (page 226) to pass them to Secure Transpport.
    c. Establish a connection using CFNetwork, BSD Sockets, or Open Transport. Then call `SSLSetConnection` (page 222) to specify the connection to which the SSL session context applies.
    d. Call `SSLSetPeerDomainName` (page 227) to specify the fully-qualified domain name of the peer to which you want to connect (optional but highly recommended).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

e. Call `SSLSetCertificate` (page 221) to specify the certificate to be used in authentication (required for server side, optional for client).

2. Starting a session
   - Call `SSLHandshake` (page 216) to perform the SSL handshake and establish a secure session.
3. Maintaining the session
   - To transfer data over the secure session, call `SSLWrite` (page 231) and `SSLRead` (page 218) as needed.
4. Ending a session
   a. Call `SSLClose` (page 202) to close the secure session.
   b. Close the connection and dispose of the connection reference (`SSLConnectionRef` (page 234)).
   c. Call `SSLDisposeContext` (page 202) to dispose of the SSL session context.
   d. If you have called `SSLGetPeerCertificates` (page 210) to obtain any certificates, call `CFRelease` to release the certificate reference objects.

In many cases, it is easier to use the CFNetwork API than Secure Transport to implement a simple connection to a secure (HTTPS) URL. See *CFNetwork Programming Guide* for documentation of the CFNetwork API and the *CFNetworkHTTPDownload* sample code for an example of code that downloads data from a URL. If you specify an HTTPS URL, this routine automatically uses Secure Transport to encrypt the data stream.

For functions to manage and evaluate certificates, see *Certificate, Key, and Trust Services Reference* and *Certificate, Key, and Trust Services Programming Guide*.

Functions by Task

Creating and Disposing of a Session Context

`SSLNewContext` (page 217)
Creates a new SSL session context.

`SSLDisposeContext` (page 202)
Disposes of an SSL session context.

Configuring an SSL Session

`SSLSetConnection` (page 222)
Specifies an I/O connection for a specific session.

`SSLGetConnection` (page 205)
Retrieves an I/O connection—such as a socket or endpoint—for a specific session.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SSLSetIOFuncs` (page 226)
Specifies callback functions that perform the network I/O operations.

`SSLSetProtocolVersionEnabled` (page 229)
Sets the allowed SSL protocol versions.

`SSLGetProtocolVersionEnabled` (page 213)
Retrieves the enabled status of a given protocol.

`SSLSetClientSideAuthenticate` (page 222)
Specifies the requirements for client-side authentication.

`SSLSetRsaBlinding` (page 230)
Enables or disables RSA blinding.

`SSLGetRsaBlinding` (page 214)
Obtains a value indicating whether RSA blinding is enabled.

Managing an SSL Session

`SSLHandshake` (page 216)
Performs the SSL handshake.

`SSLGetSessionState` (page 215)
Retrieves the state of an SSL session.

`SSLGetNegotiatedProtocolVersion` (page 208)
Obtains the negotiated protocol version of the active session.

`SSLSetPeerID` (page 228)
Specifies data that is sufficient to uniquely identify the peer of the current session.

`SSLGetPeerID` (page 212)
Retrieves the current peer ID data.

`SSLGetBufferedReadSize` (page 204)
Determines how much data is available to be read.

`SSLRead` (page 218)
Performs a normal application-level read operation.

`SSLWrite` (page 231)
Performs a normal application-level write operation.

`SSLClose` (page 202)
Terminates the current SSL session.

Managing Ciphers

`SSLGetNumberSupportedCiphers` (page 210)
Determines the number of cipher suites supported.

`SSLGetSupportedCiphers` (page 215)
Determines the values of the supported cipher suites.

`SSLSetEnabledCiphers` (page 225)
Specifies a restricted set of SSL cipher suites to be enabled by the current SSL session context.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SSLGetNumberEnabledCiphers` (page 209)
Determines the number of cipher suites currently enabled.

`SSLGetEnabledCiphers` (page 207)
Determines which SSL cipher suites are currently enabled.

`SSLGetNegotiatedCipher` (page 208)
Retrieves the cipher suite negotiated for this session.

`SSLSetDiffieHellmanParams` (page 223)
Specifies Diffie-Hellman parameters.

`SSLGetDiffieHellmanParams` (page 206)
Retrieves the Diffie-Hellman parameters specified earlier.

Managing Root Certificates

`SSLSetAllowsAnyRoot` (page 218)
Specifies whether root certificates from unrecognized certification authorities are allowed.

`SSLGetAllowsAnyRoot` (page 203)
Obtains a value specifying whether an unknown root is allowed.

`SSLSetAllowsExpiredRoots` (page 220)
Specifies whether expired root certificates are allowed.

`SSLGetAllowsExpiredRoots` (page 204)
Retrieves the value indicating whether expired roots are allowed.

`SSLSetTrustedRoots` (page 231)
Augments or replaces the default set of trusted root certificates for this session.

`SSLGetTrustedRoots` (page 216) Deprecated in Mac OS X v10.5
Retrieves the current list of trusted root certificates.

Managing Certificates

`SSLAddDistinguishedName` (page 201)
Unsupported.

`SSLSetAllowsExpiredCerts` (page 219)
Specifies whether certificate expiration times are ignored.

`SSLGetAllowsExpiredCerts` (page 203)
Retrieves the value specifying whether expired certificates are allowed.

`SSLSetCertificate` (page 221)
Specifies this connection's certificate or certificates.

`SSLGetClientCertificateState` (page 205)
Retrieves the exchange status of the client certificate.

`SSLSetEnableCertVerify` (page 224)
Enables or disables peer certificate chain validation.

`SSLGetEnableCertVerify` (page 207)
Determines whether peer certificate chain validation is currently enabled.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SSLSetEncryptionCertificate` (page 225)
Specifies the encryption certificates used for this connection.

`SSLGetPeerCertificates` (page 210) Deprecated in Mac OS X v10.5
Retrieves a peer certificate.

Managing the Peer Domain Name

`SSLSetPeerDomainName` (page 227)
Specifies the fully qualified domain name of the peer.

`SSLGetPeerDomainNameLength` (page 212)
Determines the length of a previously set peer domain name.

`SSLGetPeerDomainName` (page 211)
Retrieves the peer domain name specified previously.

Deprecated Functions

`SSLSetProtocolVersion` (page 228)
Sets the SSL protocol version. This function is deprecated.

`SSLGetProtocolVersion` (page 213)
Gets the SSL protocol version. This function is deprecated.

Functions

SSLAddDistinguishedName

Unsupported.

```
OSStatus SSLAddDistinguishedName (
   SSLContextRef context,
   const void *derDN,
   size_t derDNLen
);
```

Parameters

context
An SSL session context reference.

derDN
A pointer to a buffer containing a DER-encoded distinguished name.

derDNLen
A value of type `size_t` representing the size of the buffer pointed to by the parameter *derDN*.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function has not been implemented and is unsupported at this time.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Usupported.

Declared In
SecureTransport.h

SSLClose

Terminates the current SSL session.

```
OSStatus SSLClose (
   SSLContextRef context
);
```

Parameters

context

The SSL session context reference of the session you want to terminate.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
SecureTransport.h

SSLDisposeContext

Disposes of an SSL session context.

```
OSStatus SSLDisposeContext (
   SSLContextRef context
);
```

Parameters

context

A reference to the SSL session context to dispose.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
When you are completely finished with a secure session, you should dispose of the SSL session context in order to release the memory associated with the session.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecureTransport.h

SSLGetAllowsAnyRoot

Obtains a value specifying whether an unknown root is allowed.

```
OSStatus SSLGetAllowsAnyRoot (
   SSLContextRef context,
   Boolean *anyRoot
);
```

Parameters

context
An SSL session context reference.

anyRoot
On return, a Boolean indicating the current setting of the `anyRoot` flag.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Use the `SSLSetAllowsAnyRoot` (page 218) function to set the value of the `anyRoot` flag. The effect and meaning of this flag is described in the discussion of the `SSLSetAllowsAnyRoot` (page 218) function.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecureTransport.h

SSLGetAllowsExpiredCerts

Retrieves the value specifying whether expired certificates are allowed.

```
OSStatus SSLGetAllowsExpiredCerts (
   SSLContextRef context,
   Boolean *allowsExpired
);
```

Parameters

context
An SSL session context reference.

allowsExpired
On return, this flag is set to the value of the Boolean flag that specifies whether expired certificates are ignored. If this value is `true`, then Secure Transport does not return an error if any certificates in the certificate chain are expired.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
You can set the `allowsExpired` flag to allow the handshake to succeed even if one or more certificates in the certificate chain have expired. This function returns the current setting of this flag. Use the `SSLSetAllowsExpiredCerts` (page 219) function to set the value of the `allowsExpired` flag.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

SSLGetAllowsExpiredRoots

Retrieves the value indicating whether expired roots are allowed.

```
OSStatus SSLGetAllowsExpiredRoots (
   SSLContextRef context,
   Boolean *allowsExpired
);
```

Parameters

context
An SSL session context reference.

allowsExpired
On return, points to a Boolean value indicating whether expired roots are allowed. If this value is `true`, no errors are returned if the certificate chain ends in an expired root.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Use the `SSLSetAllowsExpiredRoots` (page 220) function to change the setting of the `allowsExpired` flag.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLGetBufferedReadSize

Determines how much data is available to be read.

```
OSStatus SSLGetBufferedReadSize (
   SSLContextRef context,
   size_t *bufSize
);
```

Parameters

context
An SSL session context reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

bufSize
On return, the size of the data to be read.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function determines how much data you can be guaranteed to obtain in a call to the `SSLRead` (page 218) function. This function does not block or cause any low-level read operations to occur.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLGetClientCertificateState

Retrieves the exchange status of the client certificate.

```
OSStatus SSLGetClientCertificateState (
   SSLContextRef context,
   SSLClientCertificateState *clientState
);
```

Parameters
context
An SSL session context reference.

clientState
On return, a pointer to a value indicating the state of the client certificate exchange. See `SSL Client Certificate State Constants` (page 237) for a list of possible values.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
The value returned reflects the latest change in the state of the client certificate exchange. If either peer initiates a renegotiation attempt, Secure Transport resets the state to `kSSLClientCertNone`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLGetConnection

Retrieves an I/O connection—such as a socket or endpoint—for a specific session.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SSLGetConnection (
   SSLContextRef context,
   SSLConnectionRef *connection
);
```

Parameters

context

An SSL session context reference.

connection

On return, a pointer to a session connection reference. If no connection has been set using the `SSLSetConnection` (page 222) function, then this parameter is `NULL` on return.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You can use this function on either the client or server to retrieve the connection associated with a secure session.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLGetDiffieHellmanParams

Retrieves the Diffie-Hellman parameters specified earlier.

```
OSStatus SSLGetDiffieHellmanParams (
   SSLContextRef context,
   const void **dhParams,
   size_t *dhParamsLen
);
```

Parameters

context

An SSL session context reference.

dhParams

On return, points to a buffer containing the Diffie-Hellman parameter block in Open SSL DER format.The returned data is not copied and belongs to the SSL session context reference; therefore, you cannot modify the data and it is released automatically when you dispose of the context.

dhParamsLen

On return, points to the length of the buffer pointed to by the *dhParams* parameter.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function returns the parameter block specified in an earlier call to the function `SSLSetDiffieHellmanParams` (page 223). If `SSLSetDiffieHellmanParams` was never called, the `dhParams` parameter returns `NULL` and the `dhParamsLen` parameter returns `0`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLGetEnableCertVerify

Determines whether peer certificate chain validation is currently enabled.

```
OSStatus SSLGetEnableCertVerify (
   SSLContextRef context,
   Boolean *enableVerify
);
```

Parameters

context
An SSL session context reference.

enableVerify
On return, a pointer to a Boolean value specifying whether peer certificate chain validation is enabled. If this value is `true`, then Secure Transport automatically attempts to verify the certificate chain during exchange of peer certificates.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Use the `SSLSetEnableCertVerify` (page 224) function to set the value of the `enableVerify` flag.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLGetEnabledCiphers

Determines which SSL cipher suites are currently enabled.

```
OSStatus SSLGetEnabledCiphers (
   SSLContextRef context,
   SSLCipherSuite *ciphers,
   size_t *numCiphers
);
```

Parameters

context
An SSL session context reference.

ciphers
On return, points to the enabled cipher suites. Before calling, you must allocate this buffer using the number of enabled cipher suites retrieved from a call to the `SSLGetNumberEnabledCiphers` (page 209) function.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

numCiphers
Pointer to the number of enabled cipher suites. Before calling, retrieve this value by calling the `SSLGetNumberEnabledCiphers` (page 209) function.

Return Value
A result code. See "Secure Transport Result Codes" (page 240). If the supplied buffer is too small, `errSSLBufferOverflow` is returned.

Discussion
Call the `SSLSetEnabledCiphers` (page 225) function to specify which SSL cipher suites are enabled.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

SSLGetNegotiatedCipher

Retrieves the cipher suite negotiated for this session.

```
OSStatus SSLGetNegotiatedCipher (
   SSLContextRef context,
   SSLCipherSuite *cipherSuite
);
```

Parameters
context
An SSL session context reference.

cipherSuite
On return, points to the cipher suite that was negotiated for this session.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You should call this function only when a session is active.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLGetNegotiatedProtocolVersion

Obtains the negotiated protocol version of the active session.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SSLGetNegotiatedProtocolVersion (
   SSLContextRef context,
   SSLProtocol *protocol
);
```

Parameters

context

An SSL session context reference.

protocol

On return, points to the negotiated protocol version of the active session.

Return Value

A result code. See "Secure Transport Result Codes" (page 240). This function returns `kSSLProtocolUnknown` if no SSL session is in progress.

Discussion

This function retrieves the version of SSL or TLS protocol negotiated for the session. Note that the negotiated protocol may not be the same as your preferred protocol, depending on which protocol versions you enabled with the `SSLSetProtocolVersionEnabled` (page 229) function. This function can return any of the following values:

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolUnknown`

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

SSLSample

Declared In

`SecureTransport.h`

SSLGetNumberEnabledCiphers

Determines the number of cipher suites currently enabled.

```
OSStatus SSLGetNumberEnabledCiphers (
   SSLContextRef context,
   size_t *numCiphers
);
```

Parameters

context

An SSL session context reference.

numCiphers

On return, points to the number of enabled cipher suites.

Return Value

A result code. See "Secure Transport Result Codes" (page 240).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
You use the number of enabled cipher suites returned by this function when you call the `SSLGetEnabledCiphers` (page 207) function to retrieve the list of currently enabled cipher suites.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

SSLGetNumberSupportedCiphers

Determines the number of cipher suites supported.

```
OSStatus SSLGetNumberSupportedCiphers (
   SSLContextRef context,
   size_t *numCiphers
);
```

Parameters

context
An SSL session context reference.

numCiphers
On return, points to the number of supported cipher suites.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You use the number of enabled cipher suites returned by this function when you call the `SSLGetNumberSupportedCiphers` (page 210) function to retrieve the list of currently enabled cipher suites.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLGetPeerCertificates

Retrieves a peer certificate. (Deprecated in Mac OS X v10.5.)

```
OSStatus SSLGetPeerCertificates (
   SSLContextRef context,
   CFArrayRef *certs
);
```

Parameters

context
An SSL session context reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

certs

On return, a pointer to an array of values of type `SecCertificateRef` representing the peer certificate and the certificate chain used to validate it. The certificate at index 0 of the returned array is the peer certificate; the root certificate (or the closest certificate to it) is at the end of the returned array. The entire array is created by the Secure Transport library; you must release it when you are finished with it.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function is valid any time after a handshake attempt. You can use it to examine a peer certificate, to examine a certificate chain to determine why a handshake attempt failed, or to retrieve the certificate chain in order to validate the certificate yourself (see `SSLSetEnableCertVerify` (page 224)).

Availability
Available in Mac OS X v10.2 and later.
Deprecated in Mac OS X v10.5.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLGetPeerDomainName

Retrieves the peer domain name specified previously.

```
OSStatus SSLGetPeerDomainName (
   SSLContextRef context,
   char *peerName,
   size_t *peerNameLen
);
```

Parameters

context

An SSL session context reference.

peerName

On return, points to the peer domain name.

peerNameLen

A pointer to the length of the peer domain name. Before calling this function, retrieve the peer domain name length by calling the function `SSLGetPeerDomainNameLength` (page 212).

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
If you previouslly called the `SSLSetPeerDomainName` (page 227) function to specify a fully qualified domain name for the peer certificate, you can use the `SSLGetPeerDomainName` function to retrieve the domain name.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Declared In
SecureTransport.h

SSLGetPeerDomainNameLength

Determines the length of a previously set peer domain name.

```
OSStatus SSLGetPeerDomainNameLength (
   SSLContextRef context,
   size_t *peerNameLen
);
```

Parameters

context
An SSL session context reference.

peerNameLen
On return, points to the length of the peer domain name.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
If you previously called the SSLSetPeerDomainName (page 227) function to specify a fully qualified domain name for the peer certificate, you can use the SSLGetPeerDomainName (page 211) function to retrieve the peer domain name. Before doing so, you must call the SSLGetPeerDomainNameLength function to retrieve the buffer size needed for the domain name.

Availability
Available in Mac OS X v10.2 and later.

Declared In
SecureTransport.h

SSLGetPeerID

Retrieves the current peer ID data.

```
OSStatus SSLGetPeerID (
   SSLContextRef context,
   const void **peerID,
   size_t *peerIDLen
);
```

Parameters

context
An SSL session context reference.

peerID
On return, points to a buffer containing the peer ID data.

peerIDLen
On return, the length of the peer ID data buffer.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
If the peer ID data for this context was not set by calling the `SSLSetPeerID` (page 228) function, this function returns a `NULL` pointer in the `peerID` parameter, and `0` in the *`peerIDLen`* parameter.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLGetProtocolVersion

Gets the SSL protocol version. This function is deprecated.

```
OSStatus SSLGetProtocolVersion (
   SSLContextRef context,
   SSLProtocol *protocol
);
```

Parameters

`context`
An SSL session context reference.

`protocol`
On return, a pointer to the SSL protocol version.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Use the `SSLGetProtocolVersionEnabled` (page 213) function instead.

Availability
Available in Mac OS X v10.2.

Declared In
`SecureTransport.h`

SSLGetProtocolVersionEnabled

Retrieves the enabled status of a given protocol.

```
OSStatus SSLGetProtocolVersionEnabled (
   SSLContextRef context,
   SSLProtocol protocol,
   Boolean *enable
);
```

Parameters

`context`
An SSL session context reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

protocol
A value of type `SSLProtocol` representing an SSL protocol version.

enable
On return, points to a Boolean value indicating whether the specified protocol version is enabled. If this value is `true`, the protocol is enabled.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You can specify any one of the following values for the `protocol` parameter:

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolAll` Specify this value to determine whether all protocols are enabled.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLGetRsaBlinding

Obtains a value indicating whether RSA blinding is enabled.

```
OSStatus SSLGetRsaBlinding (
   SSLContextRef context,
   Boolean *blinding
);
```

Parameters
context
An SSL session context reference.

blinding
On return, a pointer to a Boolean value indicating whether RSA blinding is enabled.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function is used only on the server side of a connection.

Call the `SSLSetRsaBlinding` (page 230) function to enable or disable RSA blinding.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SSLGetSessionState

Retrieves the state of an SSL session.

```
OSStatus SSLGetSessionState (
   SSLContextRef context,
   SSLSessionState *state
);
```

Parameters

context

An SSL session context reference.

state

On return, points to a constant that indicates the state of the SSL session. See "SSL Session State Constants" (page 239) for possible values.

Return Value

A result code. See "Secure Transport Result Codes" (page 240).

Availability

Available in Mac OS X v10.2 and later.

Declared In

`SecureTransport.h`

SSLGetSupportedCiphers

Determines the values of the supported cipher suites.

```
OSStatus SSLGetSupportedCiphers (
   SSLContextRef context,
   SSLCipherSuite *ciphers,
   size_t *numCiphers
);
```

Parameters

context

An SSL session context reference.

ciphers

On return, points to the values of the supported cipher suites. Before calling, you must allocate this buffer using the number of supported cipher suites retrieved from a call to the `SSLGetNumberSupportedCiphers` (page 210) function.

numCiphers

Points to the number of supported cipher suites that you want returned. Before calling, retrieve this value by calling the `SSLGetNumberSupportedCiphers` (page 210) function.

Return Value

A result code. See "Secure Transport Result Codes" (page 240). If the supplied buffer is too small, `errSSLBufferOverflow` is returned.

Discussion

All the supported cipher suites are enabled by default. Use the `SSLSetEnabledCiphers` (page 225) function to enable a subset of the supported cipher suites. Use the `SSLGetEnabledCiphers` (page 207) function to determine which cipher suites are currently enabled.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLGetTrustedRoots

Retrieves the current list of trusted root certificates. (Deprecated in Mac OS X v10.5.)

```
OSStatus SSLGetTrustedRoots (
   SSLContextRef context,
   CFArrayRef *trustedRoots
);
```

Parameters

context

An SSL session context reference.

trustedRoots

On return, a pointer to a value of type `CFArrayRef`. This array contains values of type `SecCertificateRef` representing the current set of trusted roots.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You can use the `SSLSetTrustedRoots` (page 231) function to replace or add to the set of trusted root certificates. If `SSLSetTrustedRoots` (page 231) has never been called for this session, the `SSLGetTrustedRoots` function returns the system's default set of trusted root certificates.

Availability
Available in Mac OS X v10.3 and later.
Deprecated in Mac OS X v10.5.

Declared In
`SecureTransport.h`

SSLHandshake

Performs the SSL handshake.

```
OSStatus SSLHandshake (
   SSLContextRef context
);
```

Parameters

context

An SSL session context reference.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
On successful return, the session is ready for normal secure communication using the functions `SSLRead` (page 218) and `SSLWrite` (page 231).

If it finds any problems with the peer's certificate chain, Secure Transport aborts the handshake. You can use the `SSLGetPeerCertificates` (page 210) function to see the peer's certificate chain. This function can return a wide variety of result codes, including the following:

- `errSSLUnknownRootCert`—The peer has a valid certificate chain, but the root of the chain is not a known anchor certificate.
- `errSSLNoRootCert`—The peer's certificate chain was not verifiable to a root certificate.
- `errSSLCertExpired`—The peer's certificate chain has one or more expired certificates.
- `errSSLXCertChainInvalid`—The peer has an invalid certificate chain; for example, signature verification within the chain failed, or no certificates were found.

A return value of `errSSLWouldBlock` indicates that the `SSLHandshake` function must be called again until a different result code is returned.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLNewContext

Creates a new SSL session context.

```
OSStatus SSLNewContext (
   Boolean isServer,
   SSLContextRef *contextPtr
);
```

Parameters

isServer
A Boolean value; `True` if the calling process is a server.

contextPtr
On return, points to a new SSL session context reference.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
The SSL session context is an opaque data structure that identifies a session and stores session information. You must pass this object to every other function in the Secure Transport API.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLRead

Performs a normal application-level read operation.

```
OSStatus SSLRead (
   SSLContextRef context,
   void *data,
   size_t dataLength,
   size_t *processed
);
```

Parameters

context
An SSL session context reference.

data
On return, points to the data read. You must allocate this buffer before calling the function. The size of this buffer must be equal to or greater than the value in the `dataLength` parameter.

dataLength
The amount of data you would like to read.

processed
On return, points to the number of bytes actually read.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
The `SSLRead` function might call the `SSLReadFunc` (page 232) function that you provide (see `SSLSetIOFuncs` (page 226). Because you may configure the underlying connection to operate in a nonblocking manner, a read operation might return `errSSLWouldBlock`, indicating that less data than requested was actually transferred. In this case, you should repeat the call to `SSLRead` until some other result is returned.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetAllowsAnyRoot

Specifies whether root certificates from unrecognized certification authorities are allowed.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SSLSetAllowsAnyRoot (
   SSLContextRef context,
   Boolean anyRoot
);
```

Parameters

context

An SSL session context reference.

anyRoot

A Boolean flag specifying whether root certificates from unrecognized certification authorities (CAs) are allowed. The default for this flag is `false`, specifying that roots from unrecognized CAs are not allowed.

Return Value

A result code. See "Secure Transport Result Codes" (page 240).

Discussion

The system maintains a set of root certificates signed by known, trusted root CAs. When the `anyRoot` flag is `true`, Secure Transport does not return an error if one of the following two conditions occurs:

- The peer returns a certificate chain with a root certificate, and the chain verifies to that root, but the CA for the root certificate is not one of the known, trusted root CAs. This results in an `errSSLUnknownRootCert` result code when the `anyRoot` flag is `false`.
- The peer returns a certificate chain that does not contain a root certificate, and the server can't verify the chain to one of the trusted root certificates. This results in an `errSSLNoRootCert` result code when the `anyRoot` flag is `false`.

Both of these error conditions are ignored when the `anyRoot` flag is `true`, allowing connection to a peer for which trust could not be established.

If you use this function to allow an untrusted root to be used for validation of a certificate—for example, after prompting the user for permission to do so—remember to set the `anyRoot` Boolean value back to `false`. If you don't, any random root certificate can be used for signing a certificate chain. To add a certificate to the list of trusted roots, use the `SecTrustSetAnchorCertificates` (page 71) function.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

SSLSample

Declared In

`SecureTransport.h`

SSLSetAllowsExpiredCerts

Specifies whether certificate expiration times are ignored.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SSLSetAllowsExpiredCerts (
   SSLContextRef context,
   Boolean allowsExpired
);
```

Parameters

context

An SSL session context reference.

allowsExpired

A Boolean flag representing whether the certificate expiration times are ignored. The default for this flag is `false`, meaning expired certificates result in an `errSSLCertExpired` result code.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You can use this function to allow the handshake to succeed even if one or more certificates in the certificate chain have expired. You can use the `SSLGetAllowsExpiredCerts` (page 203) function to determine the current setting of the `allowsExpired` flag.

Use the `SSLSetAllowsExpiredRoots` (page 220) function to set a flag specifying whether expired root certificates are allowed.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetAllowsExpiredRoots

Specifies whether expired root certificates are allowed.

```
OSStatus SSLSetAllowsExpiredRoots (
   SSLContextRef context,
   Boolean allowsExpired
);
```

Parameters

context

An SSL session context reference.

allowsExpired

A Boolean value indicating whether to allow expired root certificates. Pass `true` to allow expired roots.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
The default value for the `allowsExpired` flag is `false`. When this flag is `false`, Secure Transport returns an `errSSLCertExpired` result code during handshake if the root certificate is expired.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

You can use the `SSLGetAllowsExpiredRoots` (page 204) function to determine the current setting of the `allowsExpired` flag.

Use the `SSLSetAllowsExpiredCerts` (page 219) function to set a value that determines whether expired non-root certificates are allowed.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLSetCertificate

Specifies this connection's certificate or certificates.

```
OSStatus SSLSetCertificate (
   SSLContextRef context,
   CFArrayRef certRefs
);
```

Parameters

context
An SSL session context reference.

certRefs
The certificates to set. This array contains items of type `SecCertificateRef`, except for `certRefs[0]`, which is of type `SecIdentityRef`.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Setting the certificate or certificates is mandatory for server connections, but is optional for clients. Specifying a certificate for a client enables SSL client-side authentication. You must place in `certRefs[0]` a `SecIdentityRef` object that identifies the leaf certificate and its corresponding private key. Specifying a root certificate is optional; if it's not specified, the root certificate that verifies the certificate chain specified here must be present in the system wide set of trusted anchor certificates.

This function can be called only when no session is active.

Secure Transport assumes the following:

- The certificate references remain valid for the lifetime of the session.
- The identity specified in `certRefs[0]` is capable of signing.

The required capabilities of the identity specified in `certRefs[0]`, and of the optional certificate specified in the `SSLSetEncryptionCertificate` (page 225) function, are highly dependent on the application. For example, to work as a server with Netscape clients, the identity specified here must be capable of both signing and encrypting.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetClientSideAuthenticate

Specifies the requirements for client-side authentication.

```
OSStatus SSLSetClientSideAuthenticate (
   SSLContextRef context,
   SSLAuthenticate auth
);
```

Parameters

context
An SSL session context reference.

auth
A flag setting the requirements for client-side authentication. See "SSL Authentication Constants" (page 235) for possible values.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function can be called only by servers. Use of this function is optional. The default authentication requirement is `kNeverAuthenticate`. This function may be called only when no session is active.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetConnection

Specifies an I/O connection for a specific session.

```
OSStatus SSLSetConnection (
   SSLContextRef context,
   SSLConnectionRef connection
);
```

Parameters

context
An SSL session context reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

connection

An SSL session connection reference. The connection data is opaque to Secure Transport; you can set it to any value that your application can use to uniquely identify the connection in the callback functions `SSLReadFunc` (page 232) and `SSLWriteFunc` (page 233).

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You must establish a connection before creating a secure session. After calling the `SSLNewContext` (page 217) function to create an SSL session context, you call the `SSLSetConnection` function to specify the connection to which the context applies. You specify a value in the `connection` parameter that your callback routines can use to identify the connection. This value might be a pointer to a socket (if you are using the Sockets API) or an endpoint (if you are using Open Transport). For example, you might create a socket, start a connection on it, create a context reference, cast the socket to an `SSLConnectionRef`, and then pass both the context reference and connection reference to the `SSLSetConnection` function.

Note that the Sockets API is the preferred networking interface for new development.

On the client side, it's assumed that communication has been established with the desired server on this connection. On the server side, it's assumed that a connection has been established in response to an incoming client request .

This function must be called prior to the `SSLHandshake` (page 216) function; consequently, this function can be called only when no session is active.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetDiffieHellmanParams

Specifies Diffie-Hellman parameters.

```
OSStatus SSLSetDiffieHellmanParams (
   SSLContextRef context,
   const void *dhParams,
   size_t dhParamsLen
);
```

Parameters

context

An SSL session context reference.

dhParams

A pointer to a buffer containing the Diffie-Hellman parameters in Open SSL DER format.

dhParamsLen

A value representing the size of the buffer pointed to by the *dhParams* parameter.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You can use this function to specify a set of Diffie-Hellman parameters to be used by Secure Transport for a specific session. Use of this function is optional. If Diffie-Hellman ciphers are allowed, the server and client negotiate a Diffie-Hellman cipher, and this function has not been called, then Secure Transport calculates a set of process wide parameters. However, that process can take as long as 30 seconds. Diffie-Hellman ciphers are enabled by default; see `SSLSetEnabledCiphers` (page 225).

In SSL/TLS, Diffie-Hellmand parameters are always specified by the server. Therefore, this function can be called only by the server side of the connection.

You can use the `SSLGetDiffieHellmanParams` (page 206) function to retrieve Diffie-Hellman parameters specified in an earlier call to `SSLSetDiffieHellmanParams`.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLSetEnableCertVerify

Enables or disables peer certificate chain validation.

```
OSStatus SSLSetEnableCertVerify (
   SSLContextRef context,
   Boolean enableVerify
);
```

Parameters

context
An SSL session context reference.

enableVerify
A Boolean value specifying whether peer certificate chain validation is enabled. Certificate chain validation is enabled by default. Specify `false` to disable validation.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
By default, Secure Transport attempts to verify the certificate chain during an exchange of peer certificates. If you disable peer certificate chain validation, it is your responsibility to call `SSLGetPeerCertificates` (page 210) upon successful completion of the handshake and then to validate the peer certificate chain before transferring the data.

You can use the `SSLGetEnableCertVerify` (page 207) function to determine the current setting of the `enableVerify` flag.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SSLSetEnabledCiphers

Specifies a restricted set of SSL cipher suites to be enabled by the current SSL session context.

```
OSStatus SSLSetEnabledCiphers (
   SSLContextRef context,
   const SSLCipherSuite *ciphers,
   size_t numCiphers
);
```

Parameters

context

An SSL session context reference.

ciphers

A pointer to the cipher suites to enable.

numCiphers

The number of cipher suites to enable.

Return Value

A result code. See "Secure Transport Result Codes" (page 240).

Discussion

You can call this function, for example, to limit cipher suites to those that use exportable key sizes or to those supported by a particular protocol version.

This function can be called only when no session is active. The default set of enabled cipher suites is the complete set of supported cipher suites obtained by calling the `SSLGetSupportedCiphers` (page 215) function.

Call the `SSLGetEnabledCiphers` (page 207) function to determine which SSL cipher suites are currently enabled.

Availability

Available in Mac OS X v10.2 and later.

Related Sample Code

SSLSample

Declared In

`SecureTransport.h`

SSLSetEncryptionCertificate

Specifies the encryption certificates used for this connection.

```
OSStatus SSLSetEncryptionCertificate (
   SSLContextRef context,
   CFArrayRef certRefs
);
```

Parameters

context

An SSL session context reference.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

certRefs

A value of type `CFArrayRef` referring to an array of certificate references. The references are type `SecCertificateRef`, except for `certRefs[0]`, which is of type `SecIdentityRef`.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Use this function in one of the following cases:

- The leaf certificate specified in the `SSLSetCertificate` (page 221) function is not capable of encryption.
- The leaf certificate specified in the `SSLSetCertificate` (page 221) function contains a key that is too large or strong for legal encryption in this session. In this case, a weaker certificate is specified here and is used for server-initiated key exchange.

The following assumptions are made:

- The *certRefs* parameter's references remain valid for the lifetime of the connection.
- The specified `certRefs[0]` value is capable of encryption.

This function can be called only when no session is active.

SSL servers that enforce the SSL3 or TLS1 specification to the letter do not accept encryption certificates with key sizes larger than 512 bits for exportable ciphers (that is, for SSL sessions with 40-bit session keys). Therefore, if you wish to support exportable ciphers and your certificate has a key larger than 512 bits, you must specify a separate encryption certificate.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetIOFuncs

Specifies callback functions that perform the network I/O operations.

```
OSStatus SSLSetIOFuncs (
   SSLContextRef context,
   SSLReadFunc read,
   SSLWriteFunc write
);
```

Parameters

context

An SSL session context reference.

read

A pointer to your read callback function. See `SSLReadFunc` (page 232) for information on defining this function.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

write
A pointer to your write callback function. See SSLWriteFunc (page 233) for information on defining this function.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Secure Transport calls your read and write callback functions to perform network I/O. You must define these functions before calling SSLSetIOFuncs.

You must call SSLSetIOFuncs prior to calling the SSLHandshake (page 216) function. SSLSetIOFuncs cannot be called while a session is active.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
SecureTransport.h

SSLSetPeerDomainName

Specifies the fully qualified domain name of the peer.

```
OSStatus SSLSetPeerDomainName (
   SSLContextRef context,
   const char *peerName,
   size_t peerNameLen
);
```

Parameters
context
An SSL session context reference.

peerName
The fully qualified domain name of the peer—for example, store.apple.com. The name is in the form of a C string, except that NULL termination is optional.

peerNameLen
The number of bytes passed in the peerName parameter.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
You can use this function to verify the common name field in the peer's certificate. If you call this function and the common name in the certificate does not match the value you specify in the peerName parameter, then handshake fails and returns errSSLXCertChainInvalid. Use of this function is optional.

This function can be called only when no session is active.

Availability
Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Related Sample Code
SSLSample

Declared In
SecureTransport.h

SSLSetPeerID

Specifies data that is sufficient to uniquely identify the peer of the current session.

```
OSStatus SSLSetPeerID (
   SSLContextRef context,
   const void *peerID,
   size_t peerIDLen
);
```

Parameters

context
An SSL session context reference.

peerID
A pointer to a buffer containing the peer ID data to set.

peerIDLen
The length of the peer ID data buffer.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Secure Transport uses the peer ID to match the peer of an SSL session with the peer of a previous session in order to resume an interrupted session. If the peer IDs match, Secure Transport attempts to resume the session with the same parameters as used in the previous session with the same peer.

The data you provide to this function is treated as an opaque blob by Secure Transport but is compared byte for byte with previous peer ID data values set by the current application. An example of peer ID data is an IP address and port, stored in some caller-private manner. Calling this function is optional but is required if you want the session to be resumable. If you do call this function, you must call it prior to the handshake for the current session.

You can use the `SSLGetPeerID` (page 212) function to retrieve the peer ID data for the current session.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
SecureTransport.h

SSLSetProtocolVersion

Sets the SSL protocol version. This function is deprecated.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
OSStatus SSLSetProtocolVersion (
   SSLContextRef context,
   SSLProtocol version
);
```

Parameters

context

An SSL session context reference.

version

The SSL protocol version to negotiate.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Use the `SSLSetProtocolVersionEnabled` (page 229) function instead.

This function cannot be called when a session is active.

Availability
Available in Mac OS X v10.2.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

SSLSetProtocolVersionEnabled

Sets the allowed SSL protocol versions.

```
OSStatus SSLSetProtocolVersionEnabled (
   SSLContextRef context,
   SSLProtocol protocol,
   Boolean enable
);
```

Parameters

context

An SSL session context reference.

protocol

The SSL protocol version to enable. Pass `kSSLProtocolAll` to enable all protocols.

enable

A Boolean value indicating whether to enable or disable the specified protocol. Specify `true` to enable the protocol.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Discussion
Calling this function is optional. The default is that all supported protocols are enabled. When you call this function, only the specified protocol is affected. Therefore, if you call it once to disable SSL version 2 (for example), the other protocols all remain enabled. You may call this function as many times as you wish to enable and disable specific protocols. You can specify one of the following values for the `protocol` parameter:

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolAll`

This function cannot be called when a session is active.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

SSLSetRsaBlinding

Enables or disables RSA blinding.

```
OSStatus SSLSetRsaBlinding (
   SSLContextRef context,
   Boolean blinding
);
```

Parameters

context
An SSL session context reference.

blinding
A Boolean value indicating whether to enable RSA blinding. Pass `true` to enable RSA blinding.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
This function is used only on the server side of a connection.

This feature thwarts a known attack to which RSA keys are vulnerable: It is possible to guess the RSA key by timing how long it takes the server to calcuate the response to certain queries. RSA blinding adds a random calculation to each query response, thus making the attack impossible. Enabling RSA blinding is a trade-off between performance and security.

RSA blinding is enabled by default. Use the `SSLGetRsaBlinding` (page 214) function to determine the current setting.

Availability
Available in Mac OS X v10.3 and later.

Declared In
`SecureTransport.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

SSLSetTrustedRoots

Augments or replaces the default set of trusted root certificates for this session.

```
OSStatus SSLSetTrustedRoots (
   SSLContextRef context,
   CFArrayRef trustedRoots,
   Boolean replaceExisting
);
```

Parameters

context

An SSL session context reference.

trustedRoots

A reference to an array of trusted root certificates of type `SecCertificateRef`.

replaceExisting

A Boolean value indicating whether to replace or append the current trusted root certificate set. If this value is `true`, the specified root certificates become the only roots that are trusted during this session. If this value is `false`, the specified root certificates are added to the current set of trusted root certificates.

Return Value

A result code. See "Secure Transport Result Codes" (page 240).

Discussion

Each successive call to this function with the *replaceExisting* parameter set to `false` results in accumulation of additional root certificates. To see the current set of trusted root certificates, call the `SSLGetTrustedRoots` (page 216) function.

Availability

Available in Mac OS X v10.3 and later.

Declared In

`SecureTransport.h`

SSLWrite

Performs a normal application-level write operation.

```
OSStatus SSLWrite (
   SSLContextRef context,
   const void *data,
   size_t dataLength,
   size_t *processed
);
```

Parameters

context

An SSL session context reference.

data

A pointer to the buffer of data to write.

dataLength

The amount, in bytes, of data to write.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

processed
On return, the length, in bytes, of the data actually written.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
The `SSLWrite` function might call the `SSLWriteFunc` (page 233) function that you provide (see `SSLSetIOFuncs` (page 226)). Because you may configure the underlying connection to operate in a no-blocking manner, a write operation might return `errSSLWouldBlock`, indicating that less data than requested was actually transferred. In this case, you should repeat the call to `SSLWrite` until some other result is returned.

Availability
Available in Mac OS X v10.2 and later.

Related Sample Code
SSLSample

Declared In
`SecureTransport.h`

Callbacks

SSLReadFunc

Defines a pointer to a customized read function that Secure Transport calls to read data from the connection.

```
typedef OSStatus (*SSLReadFunc) (
   SSLConnectionRef connection,
   void *data,
   size_t *dataLength
);
```

You would declare your callback function like this if you were to name it `MySSLReadFunction`:

```
OSStatus MySSLReadFunction (
   SSLConnectionRef connection,
   void *data,
   size_t *dataLength
);
```

Parameters

connection
A connection reference.

data
On return, points to the data read from the connection. You must allocate this memory before calling this function.

dataLength
On input, a pointer to an integer representing the length of the data in bytes. On return, points to the number of bytes actually transferred.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Before using the Secure Transport API, you must write the functions `SSLReadFunc` and `SSLWriteFunc` (page 233) and provide them to the library by calling the `SSLSetIOFuncs` (page 226) function.

You may configure the underlying connection to operate in a nonblocking manner; in that case, a read operation may well return `errSSLWouldBlock`, indicating less data than requested was transferred and nothing is wrong except that the requested I/O hasn't completed. This result is returned to the caller from the functions `SSLRead` (page 218), `SSLWrite` (page 231), or `SSLHandshake` (page 216).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

SSLWriteFunc

Defines a pointer to a customized write function that Secure Transport calls to write data to the connection.

```
typedef OSStatus (*SSLWriteFunc) (
   SSLConnectionRef connection,
   const void *data,
   size_t *dataLength
);
```

You would declare your callback function like this if you were to name it `MySSLWriteFunction`:

```
OSStatus MySSLWriteFunction (
   SSLConnectionRef connection,
   void *data,
   size_t *dataLength
);
```

Parameters

connection
The SSL session connection reference.

data
A pointer to the data to write to the connection.You must allocate this memory before calling this function.

dataLength
Before calling, an integer representing the length of the data in bytes. On return, this is the number of bytes actually transferred.

Return Value
A result code. See "Secure Transport Result Codes" (page 240).

Discussion
Before using the Secure Transport API, you must write the functions `SSLReadFunc` (page 232) and SSLWriteFunc and provide them to the library by calling the `SSLSetIOFuncs` (page 226) function.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

You may configure the underlying connection to operate in a nonblocking manner. In that case, a write operation may well return `errSSLWouldBlock`, indicating less data than requested was transferred and nothing is wrong except that the requested I/O hasn't completed. This result is returned to the caller from the functions `SSLRead` (page 218), `SSLWrite` (page 231), or `SSLHandshake` (page 216).

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

Data Types

SSLConnectionRef

Represents a pointer to an opaque I/O connection object.

```
typedef const void *SSLConnectionRef;
```

Discussion
The I/O connection object refers to data that identifies a connection. The connection data is opaque to Secure Transport; you can set it to any value that your application can use in the callback functions `SSLReadFunc` (page 232) and `SSLWriteFunc` (page 233) to uniquely identify the connection, such as a socket or endpoint. Use the `SSLSetConnection` (page 222) function to assign a value to the connection object.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

SSLContextRef

Represents a pointer to an opaque SSL session context object.

```
struct SSLContext;
typedef struct SSLContext *SSLContextRef;
```

Discussion
The SSL session context object references the state associated with a session. You cannot reuse an SSL session context in multiple sessions.

Availability
Available in Mac OS X v10.2 and later.

Declared In
`SecureTransport.h`

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Constants

SSL Authentication Constants

Represents the requirements for client-side authentication.

```
typedef enum {
    kNeverAuthenticate,
    kAlwaysAuthenticate,
    kTryAuthenticate
} SSLAuthenticate;
```

Constants

`kNeverAuthenticate`

Indicates that client-side authentication is not required. (Default.)

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kAlwaysAuthenticate`

Indicates that client-side authentication is required.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kTryAuthenticate`

Indicates that client-side authentication should be attempted. There is no error if the client doesn't have a certificate.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

SSL Cipher Suite Constants

Represents the cipher suites available.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef UInt32 SSLCipherSuite;
enum
{SSL_NULL_WITH_NULL_NULL =                  0x0000,
    SSL_RSA_WITH_NULL_MD5 =                 0x0001,
    SSL_RSA_WITH_NULL_SHA =                 0x0002,
    SSL_RSA_EXPORT_WITH_RC4_40_MD5 =        0x0003,
    SSL_RSA_WITH_RC4_128_MD5 =              0x0004,
    SSL_RSA_WITH_RC4_128_SHA =              0x0005,
    SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5 =    0x0006,
    SSL_RSA_WITH_IDEA_CBC_SHA =             0x0007,
    SSL_RSA_EXPORT_WITH_DES40_CBC_SHA =     0x0008,
    SSL_RSA_WITH_DES_CBC_SHA =              0x0009,
    SSL_RSA_WITH_3DES_EDE_CBC_SHA =         0x000A,
    SSL_DH_DSS_EXPORT_WITH_DES40_CBC_SHA =  0x000B,
    SSL_DH_DSS_WITH_DES_CBC_SHA =           0x000C,
    SSL_DH_DSS_WITH_3DES_EDE_CBC_SHA =      0x000D,
    SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA =  0x000E,
    SSL_DH_RSA_WITH_DES_CBC_SHA =           0x000F,
    SSL_DH_RSA_WITH_3DES_EDE_CBC_SHA =      0x0010,
    SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA = 0x0011,
    SSL_DHE_DSS_WITH_DES_CBC_SHA =          0x0012,
    SSL_DHE_DSS_WITH_3DES_EDE_CBC_SHA =     0x0013,
    SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA = 0x0014,
    SSL_DHE_RSA_WITH_DES_CBC_SHA =          0x0015,
    SSL_DHE_RSA_WITH_3DES_EDE_CBC_SHA =     0x0016,
    SSL_DH_anon_EXPORT_WITH_RC4_40_MD5 =    0x0017,
    SSL_DH_anon_WITH_RC4_128_MD5 =          0x0018,
    SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA = 0x0019,
    SSL_DH_anon_WITH_DES_CBC_SHA =          0x001A,
    SSL_DH_anon_WITH_3DES_EDE_CBC_SHA =     0x001B,
    SSL_FORTEZZA_DMS_WITH_NULL_SHA =        0x001C,
    SSL_FORTEZZA_DMS_WITH_FORTEZZA_CBC_SHA =0x001D,
    SSL_RSA_WITH_RC2_CBC_MD5 =              0xFF80,
    SSL_RSA_WITH_IDEA_CBC_MD5 =             0xFF81,
    SSL_RSA_WITH_DES_CBC_MD5 =              0xFF82,
    SSL_RSA_WITH_3DES_EDE_CBC_MD5 =         0xFF83,
    SSL_NO_SUCH_CIPHERSUITE =               0xFFFF
};
```

Constants

`SSL_RSA_EXPORT_WITH_RC4_40_MD5`

Session key size conforms to pre-1998 US export restrictions.

Available in Mac OS X v10.2 and later.

Declared in `CipherSuite.h`.

`SSL_RSA_EXPORT_WITH_RC2_CBC_40_MD5`

Session key size conforms to pre-1998 US export restrictions.

Available in Mac OS X v10.2 and later.

Declared in `CipherSuite.h`.

`SSL_RSA_EXPORT_WITH_DES40_CBC_SHA`

Session key size conforms to pre-1998 US export restrictions.

Available in Mac OS X v10.2 and later.

Declared in `CipherSuite.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`SSL_DH_RSA_EXPORT_WITH_DES40_CBC_SHA`
Session key size conforms to pre-1998 US export restrictions.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_DHE_DSS_EXPORT_WITH_DES40_CBC_SHA`
Session key size conforms to pre-1998 US export restrictions.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_DHE_RSA_EXPORT_WITH_DES40_CBC_SHA`
Session key size conforms to pre-1998 US export restrictions.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_DH_anon_EXPORT_WITH_RC4_40_MD5`
Session key size conforms to pre-1998 US export restrictions.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_DH_anon_EXPORT_WITH_DES40_CBC_SHA`
Session key size conforms to pre-1998 US export restrictions.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_RSA_WITH_RC2_CBC_MD5`
This value can be specified for SSL 2 but not SSL 3.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_RSA_WITH_IDEA_CBC_MD5`
This value can be specified for SSL 2 but not SSL 3.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_RSA_WITH_DES_CBC_MD5`
This value can be specified for SSL 2 but not SSL 3.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

`SSL_RSA_WITH_3DES_EDE_CBC_MD5`
This value can be specified for SSL 2 but not SSL 3.
Available in Mac OS X v10.2 and later.
Declared in `CipherSuite.h`.

SSL Client Certificate State Constants

Represents the status of client certificate exchange.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef enum {
    kSSLClientCertNone,
    kSSLClientCertRequested,
    kSSLClientCertSent,
    kSSLClientCertRejected
} SSLClientCertificateState;
```

Constants

`kSSLClientCertNone`
Indicates that the server hasn't asked for a certificate and that the client hasn't sent one.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLClientCertRequested`
Indicates that the server has asked for a certificat, but the client has not sent it.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLClientCertSent`
Indicates that the server asked for a certificate, the client sent one, and the server validated it. The application can inspect the certificate using the function `SSLGetPeerCertificates` (page 210).

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLClientCertRejected`
Indicates that the client sent a certificate but the certificate failed validation. This value is seen only on the server side. The server application can inspect the certificate using the function `SSLGetPeerCertificates` (page 210).

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

SSL Protocol Constants

Represents the SSL protocol version.

```
typedef enum {
    kSSLProtocolUnknown,
    kSSLProtocol2,
    kSSLProtocol3,
    kSSLProtocol3Only,
    kTLSProtocol1,
    kTLSProtocol1Only,
    kSSLProtocolAll
} SSLProtocol;
```

Constants

`kSSLProtocolUnknown`
Specifies that no protocol has been or should be negotiated or specified; use default.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

`kSSLProtocol2`
Specifies that only the SSL 2.0 protocol may be negotiated.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocol3`
Specifies that the SSL 3.0 protocol is preferred; the SSL 2.0 protocol may be negotiated if the peer cannot use the SSL 3.0 protocol.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocol3Only`
Specifies that only the SSL 3.0 protocol may be negotiated; fails if the peer tries to negotiate the SSL 2.0 protocol.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kTLSProtocol1`
Specifies that the TLS 1.0 protocol is preferred but lower versions may be negotiated.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kTLSProtocol1Only`
Specifies that only the TLS 1.0 protocol may be negotiated.

Available in Mac OS X v10.2 and later.

Declared in `SecureTransport.h`.

`kSSLProtocolAll`
Specifies all supported versions.

Available in Mac OS X v10.3 and later.

Declared in `SecureTransport.h`.

Discussion
The descriptions given here apply to the functions `SSLSetProtocolVersion` (page 228) and `SSLGetProtocolVersion` (page 213). For the functions `SSLSetProtocolVersionEnabled` (page 229) and `SSLGetProtocolVersionEnabled` (page 213), only the following values are used. For these functions, each constant except `kSSLProtocolAll` specifies a single protocol version.

- `kSSLProtocol2`
- `kSSLProtocol3`
- `kTLSProtocol1`
- `kSSLProtocolAll`

SSL Session State Constants

Represents the state of an SSL session.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

```
typedef enum {
    kSSLIdle,
    kSSLHandshake,
    kSSLConnected,
    kSSLClosed,
    kSSLAborted
} SSLSessionState;
```

Constants

`kSSLIdle`
No I/O has been performed yet.
Available in Mac OS X v10.2 and later.
Declared in `SecureTransport.h`.

`kSSLHandshake`
The SSL handshake is in progress.
Available in Mac OS X v10.2 and later.
Declared in `SecureTransport.h`.

`kSSLConnected`
The SSL handshake is complete; the connection is ready for normal I/O.
Available in Mac OS X v10.2 and later.
Declared in `SecureTransport.h`.

`kSSLClosed`
The connection closed normally.
Available in Mac OS X v10.2 and later.
Declared in `SecureTransport.h`.

`kSSLAborted`
The connection aborted.
Available in Mac OS X v10.2 and later.
Declared in `SecureTransport.h`.

Result Codes

The most common result codes returned by Secure Transport functions are listed in the table below.

Errors in the range of –9819 through –9840 are fatal errors that are detected by the peer.

Result Code	Value	Description
`errSSLProtocol`	–9800	SSL protocol error. Available in Mac OS X v10.2 and later.
`errSSLNegotiation`	–9801	The cipher suite negotiation failed. Available in Mac OS X v10.2 and later.
`errSSLFatalAlert`	–9802	A fatal alert was encountered. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSSLWouldBlock	–9803	Function is blocked; waiting for I/O. This is not fatal. Available in Mac OS X v10.2 and later.
errSSLSessionNotFound	–9804	An attempt to restore an unknown session failed. Available in Mac OS X v10.2 and later.
errSSLClosedGraceful	–9805	The connection closed gracefully. Available in Mac OS X v10.2 and later.
errSSLClosedAbort	–9806	The connection closed due to an error. Available in Mac OS X v10.2 and later.
errSSLXCertChainInvalid	–9807	Invalid certificate chain. Available in Mac OS X v10.2 and later.
errSSLBadCert	–9808	Bad certificate format. Available in Mac OS X v10.2 and later.
errSSLCrypto	–9809	An underlying cryptographic error was encountered. Available in Mac OS X v10.2 and later.
errSSLInternal	–9810	Internal error. Available in Mac OS X v10.2 and later.
errSSLModuleAttach	–9811	Module attach failure. Available in Mac OS X v10.2 and later.
errSSLUnknownRootCert	–9812	Certificate chain is valid, but root is not trusted. Available in Mac OS X v10.2 and later.
errSSLNoRootCert	–9813	No root certificate for the certificate chain. Available in Mac OS X v10.2 and later.
errSSLCertExpired	–9814	The certificate chain had an expired certificate. Available in Mac OS X v10.2 and later.
errSSLCertNotYetValid	–9815	The certificate chain had a certificate that is not yet valid. Available in Mac OS X v10.2 and later.
errSSLClosedNoNotify	–9816	The server closed the session with no notification. Available in Mac OS X v10.2 and later.
errSSLBufferOverflow	–9817	An insufficient buffer was provided. Available in Mac OS X v10.2 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSSLBadCipherSuite	–9818	A bad SSL cipher suite was encountered. Available in Mac OS X v10.2 and later.
errSSLPeerUnexpectedMsg	–9819	An unexpected message was received. Available in Mac OS X v10.3 and later.
errSSLPeerBadRecordMac	–9820	A bad record MAC was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerDecryptionFail	–9821	Decryption failed. Available in Mac OS X v10.3 and later.
errSSLPeerRecordOverflow	–9822	A record overflow occurred. Available in Mac OS X v10.3 and later.
errSSLPeerDecompressFail	–9823	Decompression failed. Available in Mac OS X v10.3 and later.
errSSLPeerHandshakeFail	–9824	The handshake failed. Available in Mac OS X v10.3 and later.
errSSLPeerBadCert	–9825	A bad certificate was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerUnsupportedCert	–9826	An unsupported certificate format was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerCertRevoked	–9827	The certificate was revoked. Available in Mac OS X v10.3 and later.
errSSLPeerCertExpired	–9828	The certificate expired. Available in Mac OS X v10.3 and later.
errSSLPeerCertUnknown	–9829	The certificate is unknown. Available in Mac OS X v10.3 and later.
errSSLIllegalParam	–9830	An illegal parameter was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerUnknownCA	–9831	An unknown certificate authority was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerAccessDenied	–9832	Access was denied. Available in Mac OS X v10.3 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Result Code	Value	Description
errSSLPeerDecodeError	–9833	A decoding error occurred. Available in Mac OS X v10.3 and later.
errSSLPeerDecryptError	–9834	A decryption error occurred. Available in Mac OS X v10.3 and later.
errSSLPeerExportRestriction	–9835	An export restriction occurred. Available in Mac OS X v10.3 and later.
errSSLPeerProtocolVersion	–9836	A bad protocol version was encountered. Available in Mac OS X v10.3 and later.
errSSLPeerInsufficientSecurity	–9837	There is insufficient security for this operation. Available in Mac OS X v10.3 and later.
errSSLPeerInternalError	–9838	An internal error occurred. Available in Mac OS X v10.3 and later.
errSSLPeerUserCancelled	–9839	The user canceled the operation. Available in Mac OS X v10.3 and later.
errSSLPeerNoRenegotiation	–9840	No renegotiation is allowed. Available in Mac OS X v10.3 and later.
errSSLConnectionRefused	–9844	The peer dropped the connection before responding. Available in Mac OS X v10.4 and later.
errSSLDecryptionFail	–9845	Decryption failed. Available in Mac OS X v10.3 and later.
errSSLBadRecordMac	–9846	A bad record MAC was encountered. Available in Mac OS X v10.3 and later.
errSSLRecordOverflow	–9847	A record overflow occurred. Available in Mac OS X v10.3 and later.
errSSLBadConfiguration	–9848	A configuration error occurred. Available in Mac OS X v10.3 and later.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Document Revision History

This table describes the changes to *Security Framework Reference*.

Date	Notes
2008-03-12	Added Randomization Services.
2006-05-23	First publication of this content as a collection of previously published documents.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Document Revision History

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

Index

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

I

K

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

N

P

S

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.

T

2008-03-12 | © 2008 Apple Inc. All Rights Reserved.