Security Framework Reference

Security

2008-03-12

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

Contents

Introduction

Framework	/System/Library/Frameworks/Security.framework
Header file directories	/System/Library/Frameworks/Security.framework/Headers
Companion guide	Secure Coding Guide
Declared in	SecBase.h SecCertificate.h SecIdentity.h SecImportExport.h SecItem.h SecKey.h SecPolicy.h SecRandom.h SecTrust.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.

Introduction

Managers

PART I

Managers

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 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

Concurrency Considerations

On iOS, all the functions in this API are thread-safe and reentrant.

On Mac OS X v10.6, some functions can block while waiting for input from the user (for example, when the user is asked to unlock a keychain or give permission to change trust settings). In general, it is safe to use the functions in this API from threads other than your main thread, but you should avoid calling the function from multiple operations, work queues, or threads concurrently. Instead, function calls should be serialized (or confined to a single thread) to prevent any potential problems. Exceptions are noted in the discussions of the relevant functions.

Functions by Task

Getting Type Identifiers

`SecCertificateGetTypeID` (page 14)
Returns the unique identifier of the opaque type to which a `SecCertificate` object belongs.

`SecIdentityGetTypeID` (page 15)
Returns the unique identifier of the opaque type to which a `SecIdentity` object belongs.

`SecKeyGetTypeID` (page 19)
Returns the unique identifier of the opaque type to which a `SecKey` object belongs.

`SecPolicyGetTypeID` (page 23)
Returns the unique identifier of the opaque type to which a `SecPolicy` object belongs.

`SecTrustGetTypeID` (page 28)
Returns the unique identifier of the opaque type to which a `SecTrust` object belongs.

Managing Certificates

`SecCertificateCreateWithData` (page 13)
Creates a certificate object from a DER representation of a certificate.

`SecCertificateCopyData` (page 12)
Returns a DER representation of a certificate given a certificate object.

`SecCertificateCopySubjectSummary` (page 12)
Returns a human-readable summary of a certificate.

Managing Identities

`SecPKCS12Import` (page 21)
Returns the identities and certificates in a PKCS #12-formatted blob.

`SecIdentityCopyCertificate` (page 14)
Retrieves a certificate associated with an identity.

`SecIdentityCopyPrivateKey` (page 15)
Retrieves the private key associated with an identity.

Cryptography and Digital Signatures

`SecKeyGeneratePair` (page 17)
Creates an asymmetric key pair.

`SecKeyRawSign` (page 20)
Generates a digital signature for a block of data.

`SecKeyRawVerify` (page 21)
Verifies a digital signature.

`SecKeyEncrypt` (page 16)
Encrypts a block of plaintext.

`SecKeyDecrypt` (page 16)
Decrypts a block of ciphertext.

`SecKeyGetBlockSize` (page 19)
Gets the block length associated with a cryptographic key.

Managing Policies

`SecPolicyCreateBasicX509` (page 22)
Returns a policy object for the default X.509 policy.

`SecPolicyCreateSSL` (page 23)
Returns a policy object for evaluating SSL certificate chains.

Managing Trust

`SecTrustCopyExceptions` (page 23)
Returns an opaque cookie containing exceptions to trust policies that will allow future evaluations of the current certificate to succeed.

`SecTrustSetExceptions` (page 31)
Sets a list of exceptions that should be ignored when evaluating the certificate.

`SecTrustCreateWithCertificates` (page 25)
Creates a trust management object based on certificates and policies.

`SecTrustEvaluate` (page 26)
Evaluates trust for the specified certificate and policies.

`SecTrustSetAnchorCertificates` (page 29)
Sets the anchor certificates used when evaluating a trust management object.

`SecTrustSetAnchorCertificatesOnly` (page 31)
Reenables trusting built-in anchor certificates.

`SecTrustSetVerifyDate` (page 32)
Sets the date and time against which the certificates in a trust management object are verified.

`SecTrustGetVerifyTime` (page 29)
Gets the absolute time against which the certificates in a trust management object are verified.

`SecTrustCopyPublicKey` (page 24)
Returns the public key for a leaf certificate after it has been evaluated.

`SecTrustGetCertificateCount` (page 28)
Returns the number of certificates in an evaluated certificate chain.

`SecTrustGetCertificateAtIndex` (page 27)
Returns a specific certificate from the certificate chain used to evaluate trust.

Functions

SecCertificateCopyData

Returns a DER representation of a certificate given a certificate object.

```
CFDataRef SecCertificateCopyData (
   SecCertificateRef certificate
);
```

Parameters

certificate
The certificate object for which you wish to return the DER (Distinguished Encoding Rules) representation of the X.509 certificate.

Return Value

The DER representation of the certificate. Call the `CFRelease` function to release this object when you are finished with it. Returns `NULL` if the data passed in the `certificate` parameter is not a valid certificate object.

Availability

Available in iOS 2.0 and later.

See Also

`SecCertificateCreateWithData` (page 13)

Declared In

`SecCertificate.h`

SecCertificateCopySubjectSummary

Returns a human-readable summary of a certificate.

```
CFStringRef SecCertificateCopySubjectSummary (
   SecCertificateRef certificate
);
```

Parameters

certificate

The certificate object for which you wish to return a summary string.

Return Value

A string that contains a human-readable summary of the contents of the certificate. Call the `CFRelease` function to release this object when you are finished with it. Returns `NULL` if the data passed in the `certificate` parameter is not a valid certificate object.

Discussion

Because all the data in the string comes from the certificate, the string is in whatever language is used in the certificate.

Availability

Available in iOS 2.0 and later.

See Also

`SecCertificateCreateWithData` (page 13)

Declared In

`SecCertificate.h`

SecCertificateCreateWithData

Creates a certificate object from a DER representation of a certificate.

```
SecCertificateRef SecCertificateCreateWithData (
   CFAllocatorRef allocator,
   CFDataRef data
);
```

Parameters

allocator

The `CFAllocator` object you wish to use to allocate the certificate object. Pass `NULL` to use the default allocator.

data

A DER (Distinguished Encoding Rules) representation of an X.509 certificate.

Return Value

The newly created certificate object. Call the `CFRelease` function to release this object when you are finished with it. Returns `NULL` if the data passed in the `data` parameter is not a valid DER-encoded X.509 certificate.

Discussion

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

Availability

Available in iOS 2.0 and later.

See Also

`SecCertificateCopyData` (page 12)

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 33) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a SecCertificateRef (page 33) 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 iOS 2.0 and later.

Declared In
SecCertificate.h

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. 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 39).

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 iOS 2.0 and later.

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. 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 39).

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

Availability
Available in iOS 2.0 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 33) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a SecIdentityRef (page 33) 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 iOS 2.0 and later.

Declared In
SecIdentity.h

SecKeyDecrypt

Decrypts a block of ciphertext.

```
OSStatus SecKeyDecrypt (
   SecKeyRef key,
   SecPadding padding,
   const uint8_t *cipherText,
   size_t cipherTextLen,
   uint8_t *plainText,
   size_t *plainTextLen
);
```

Parameters

key

private key with which to decrypt the data.

padding

The type of padding used. Possible values are listed in "Digital Signature Padding Types" (page 34). Typically, `kSecPaddingPKCS1` is used, which removes PKCS1 padding after decryption. If you specify `kSecPaddingNone`, the decrypted data is returned as-is.

cipherText

The data to decrypt.

cipherTextLen

Length in bytes of the data in the `cipherText` buffer. This must be less than or equal to the value returned by the `SecKeyGetBlockSize` function.

plainText

On return, the decrypted text.

plainTextLen

On input, the size of the buffer provided in the `plainText` parameter. On output, the amount of data actually placed in the buffer.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 39).

Discussion

The input buffer (`cipherText`) can be the same as the output buffer (`plainText`) to reduce the amount of memory used by the function.

Availability

Available in iOS 2.0 and later.

Related Sample Code

CryptoExercise

Declared In

`SecKey.h`

SecKeyEncrypt

Encrypts a block of plaintext.

```
OSStatus SecKeyEncrypt (
   SecKeyRef key,
   SecPadding padding,
   const uint8_t *plainText,
   size_t plainTextLen,
   uint8_t *cipherText,
   size_t *cipherTextLen
);
```

Parameters

key

Public key with which to encrypt the data.

padding

The type of padding to use. Possible values are listed in "Digital Signature Padding Types" (page 34). Typically, `kSecPaddingPKCS1` is used, which adds PKCS1 padding before encryption. If you specify `kSecPaddingNone`, the data is encrypted as-is.

plainText

The data to encrypt.

plainTextLen

Length in bytes of the data in the `plainText` buffer. This must be less than or equal to the value returned by the `SecKeyGetBlockSize` function. When PKCS1 padding is performed, the maximum length of data that can be encrypted is 11 bytes less than the value returned by the `SecKeyGetBlockSize` function (`secKeyGetBlockSize() - 11`).

cipherText

On return, the encrypted text.

cipherTextLen

On input, the size of the buffer provided in the `cipherText` parameter. On output, the amount of data actually placed in the buffer.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 39).

Discussion
The input buffer (`plainText`) can be the same as the output buffer (`cipherText`) to reduce the amount of memory used by the function.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

Declared In
`SecKey.h`

SecKeyGeneratePair

Creates an asymmetric key pair.

```
OSStatus SecKeyGeneratePair (
   CFDictionaryRef parameters,
   SecKeyRef *publicKey,
   SecKeyRef *privateKey
);
```

Parameters

parameters

A dictionary of key-value pairs that specify the type of keys to be generated.

publicKey

On return, points to the keychain item object of the new public 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. 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 39).

Discussion

In order to generate a key pair, the dictionary passed in the `parameters` parameter must contain at least the following key-value pairs:

- A `kSecAttrKeyType` key with a value of any key type defined in `SecItem.h` (see *Keychain Services Reference*), for example, `kSecAttrKeyTypeRSA`.
- A `kSecAttrKeySizeInBits` key with a value specifying the requested key size in bits. This can be specified as either a `CFNumberRef` or `CFStringRef` value. For example, RSA keys may have key size values of 512, 768, 1024, or 2048.

In addition, you can specify a number of attributes for the public and private keys individually. You can do so either by adding key-value pairs to the dictionary directly, or by adding either or both of the keys `kSecPrivateKeyAttrs` and `kSecPublicKeyAttrs`. Each of these keys takes as a value a dictionary of key-value pairs that you can use to set these attributes. The possible attributes are as follows; for details on each attribute, see *Keychain Services Reference*:

- `kSecAttrLabel` Default `NULL`.
- `kSecAttrIsPermanent` If this key is present and has a Boolean value of `true`, the key or key pair is added to the default keychain.
- `kSecAttrApplicationTag` Default `NULL`.
- `kSecAttrEffectiveKeySize` Default (`NULL`) sets the effective key size to the same as the total key size (`kSecAttrKeySizeInBits`).
- `kSecAttrCanEncrypt` Default `false` for private keys, `true` for public keys.
- `kSecAttrCanDecrypt` Default `true` for private keys, `false` for public keys.
- `kSecAttrCanDerive` Default `true`.
- `kSecAttrCanSign` Default `true` for private keys, `false` for public keys.
- `kSecAttrCanVerify` Default `false` for private keys, `true` for public keys.

- `kSecAttrCanUnwrap` Default `true` for private keys, `false` for public keys.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

Declared In
`SecKey.h`

SecKeyGetBlockSize

Gets the block length associated with a cryptographic key.

```
size_t SecKeyGetBlockSize (
   SecKeyRef key
);
```

Parameters

key
The key for which you want the block length.

Return Value
The block length associated with the key in bytes. If the key is an RSA key, for example, this is the size of the modulus.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

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 34) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecKeyRef` (page 34) 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 iOS 2.0 and later.

Declared In
`SecKey.h`

SecKeyRawSign

Generates a digital signature for a block of data.

```
OSStatus SecKeyRawSign (
   SecKeyRef key,
   SecPadding padding,
   const uint8_t *dataToSign,
   size_t dataToSignLen,
   uint8_t *sig,
   size_t *sigLen
);
```

Parameters

key
Private key with which to sign the data.

padding
The type of padding to use. Possible values are listed in "Digital Signature Padding Types" (page 34). Use `kSecPaddingPKCS1SHA1` if the data to be signed is a SHA1 digest of the actual data. If you specify `kSecPaddingNone`, the data is signed as-is.

dataToSign
The data to be signed. Typically, a digest of the actual data is signed.

dataToSignLen
Length in bytes of the data in the `dataToSign` buffer. When PKCS1 padding is performed, the maximum length of data that can be signed is 11 bytes less than the value returned by the `SecKeyGetBlockSize` function (`secKeyGetBlockSize() - 11`).

sig
On return, the digital signature.

sigLen
On input, the size of the buffer provided in the `sig` parameter. On output, the amount of data actually placed in the buffer.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 39).

Discussion
The behavior this function with `kSecPaddingNone` is undefined if the first byte of the data to sign is `0`; there is no way to verify leading zeroes, as they are discarded during the calculation.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

Declared In
SecKey.h

SecKeyRawVerify

Verifies a digital signature.

```
OSStatus SecKeyRawVerify (
   SecKeyRef key,
   SecPadding padding,
   const uint8_t *signedData,
   size_t signedDataLen,
   const uint8_t *sig,
   size_t sigLen
);
```

Parameters

key
Public key with which to verify the data.

padding
The type of padding used. Possible values are listed in "Digital Signature Padding Types" (page 34). Use `kSecPaddingPKCS1SHA1` if you are verifying a PKCS1-style signature with DER encoding of the digest type and the signed data is a SHA1 digest of the actual data. Specify `kSecPaddingNone` if no padding was used.

signedData
The data for which the signature is being verified. Typically, a digest of the actual data is signed.

signedDataLen
Length in bytes of the data in the `signedData` buffer.

sig
The digital signature to be verified.

sigLen
Length of the data in the `sig` buffer.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 39).

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

Declared In
SecKey.h

SecPKCS12Import

Returns the identities and certificates in a PKCS #12-formatted blob.

```
OSStatus SecPKCS12Import(
   CFDataRef pkcs12_data,
   CFDictionaryRef options,
   CFArrayRef *items
);
```

Parameters

pkcs12_data

The PKCS #12 data you wish to decode.

options

A dictionary of key-value pairs specifying options for the function.

items

On return, an array of `CFDictionary` key-value dictionaries. The function returns one dictionary for each item (identity or certificate) in the PKCS #12 blob. For a list of dictionary keys, see "PKCS #12 Import Item Keys" (page 38).

Return Value

A result code. The function returns `errSecSuccess` if there were no errors, `errSecDecode` if the blob can't be read or is malformed, and `errSecAuthFailed` if the password was not correct or data in the blob was damaged. See "Certificate, Key, and Trust Services Result Codes" (page 39).

Discussion

Your application can import a PKCS #12–formatted blob (a file with extension `.p12`) containing certificates and identities, where an identity is a digital certificate together with its associated private key. You can use the `SecPKCS12Import` function to obtain `SecIdentityRef` objects (including `SecCertificateRef` and `SecKeyRef` objects) for the identities in the blob, together with `SecCertificateRef` objects for the certificates in the blob needed to validate the identity, and `SecTrustRef` trust management objects needed to evaluate trust for the identities. You can then use the Keychain Services API (see *Keychain Services Reference*) to put the identities and associated certificates in the keychain.

Availability

Available in iOS 2.0 and later.

Declared In

`SecImportExport.h`

SecPolicyCreateBasicX509

Returns a policy object for the default X.509 policy.

```
SecPolicyRef SecPolicyCreateBasicX509 (
   void
);
```

Return Value

The policy object. Call the `CFRelease` function to release the object when you are finished with it.

Availability

Available in iOS 2.0 and later.

Declared In

`SecPolicy.h`

SecPolicyCreateSSL

Returns a policy object for evaluating SSL certificate chains.

```
SecPolicyRef SecPolicyCreateSSL (
   Boolean server,
   CFStringRef hostname
);
```

Parameters

server

Specify `true` to return a policy for SSL server certificates.

hostname

If you specify a value for this parameter, the policy will require the specified value to match the host name in the leaf certificate.

Return Value

The policy object. Call the `CFRelease` function to release the object when you are finished with it.

Availability

Available in iOS 2.0 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 34) object.

Discussion

This function returns a value that uniquely identifies the opaque type of a `SecPolicyRef` (page 34) 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 iOS 2.0 and later.

Declared In

`SecPolicy.h`

SecTrustCopyExceptions

Returns an opaque cookie containing exceptions to trust policies that will allow future evaluations of the current certificate to succeed.

```
CFDataRef SecTrustCopyExceptions(
    SecTrustRef trust
);
```

Parameters

trust

The evaluated trust management object whose policies you wish to retrieve.

Return Value

An opaque cookie. If you pass this cookie to `SecTrustSetExceptions` (page 31), that function sets a list of exceptions for future processing of the certificate. Once this list of exceptions are set, a subsequent call to `SecTrustEvaluate` (page 26) for that certificate will return `kSecTrustResultProceed` (page 36).

Note: If a new error occurs that did not occur when this function was called originally, the subsequent call to `SecTrustEvaluate` (page 26) can still fail. For example, if the certificate expires between calling `SecTrustCopyExceptions` and `SecTrustEvaluate` (page 26), evaluation will fail.

Discussion

Normally this API should only be called after asking the user how to proceed, and even then, only if the user explicitly tells your application to trust the current certificate chain in spite of the errors presented.

Availability

Available in iOS 4.0 and later.

Declared In

`SecTrust.h`

SecTrustCopyPublicKey

Returns the public key for a leaf certificate after it has been evaluated.

```
SecKeyRef SecTrustCopyPublicKey (
   SecTrustRef trust
);
```

Parameters

trust

The trust management object for the certificate that has been evaluated. Use the `SecTrustCreateWithCertificates` (page 25) function to create a trust management object.

Return Value

The leaf certificate's public key, or `NULL` if it the public key could not be extracted (this can happen with DSA certificate chains if the parameters in the chain cannot be found). Call the `CFRelease` function to release this object when you are finished with it.

Discussion

You must call the `SecTrustEvaluate` (page 26) function before calling this function. When you call this function, it attempts to return the public key of the leaf certificate, even if the trust evaluation was unsuccessful. Even if the trust evaluation was successful, this function might still return `NULL`—for example, if the leaf certificate's key can't be extracted for some reason.

Availability

Available in iOS 2.0 and later.

Declared In
SecTrust.h

SecTrustCreateWithCertificates

Creates a trust management object based on certificates and policies.

```
OSStatus SecTrustCreateWithCertificates (
   CFTypeRef 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.

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 function (not available on iOS) 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 39).

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 26) 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) and in the system's store of anchor certificates (see SecTrustSetAnchorCertificates (page 29)). However, you should gain a significant performance benefit by passing in the entire certificate chain, in order, in the certificates parameter.

Availability
Available in iOS 2.0 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 25) 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 36) for descriptions of possible values.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 39).

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` or `SecTrustSetVerifyDate` (page 32)) 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 `SecTrustSettingsSetTrustSettings` and `SecTrustSettingsCopyTrustSettings`). 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` 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`) and in the system's store of anchor certificates (see `SecTrustSetAnchorCertificates` (page 29)).

By default, `SecTrustEvaluate` uses the current date and time when verifying a certificate. However, you can call the `SecTrustSetVerifyDate` (page 32) 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` 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` function to get more information about the results of the trust evaluation, or the `SecTrustGetCssmResult` function to get information about the evaluation in a form that can be passed to CSSM functions.

Special Considerations

It is not safe to call this function concurrently with any other function that uses the same trust management object, or to re-enter this function for the same trust management object.

Because this function might look on the network for certificates in the certificate chain, the function might block while attempting network access.

Availability

Available in iOS 2.0 and later.

Declared In

`SecTrust.h`

SecTrustGetCertificateAtIndex

Returns a specific certificate from the certificate chain used to evaluate trust.

```
SecCertificateRef SecTrustGetCertificateAtIndex (
   SecTrustRef trust,
   CFIndex ix
);
```

Parameters

trust

The trust management object for the certificate that has been evaluated. Use the `SecTrustCreateWithCertificates` (page 25) function to create a trust management object and the `SecTrustEvaluate` (page 26) function to evaluate the certificate chain.

ix

The index number of the requested certificate. Index numbers start at 0 for the leaf certificate and end at the anchor (or the last certificate if no anchor was found). Use the `SecTrustGetCertificateCount` (page 28) function to get the total number of certificates in the chain.

Return Value

A certificate object for the requested certificate.

Discussion

You must call the `SecTrustEvaluate` (page 26) function before calling this function.

Availability
Available in iOS 2.0 and later.

Declared In
SecTrust.h

SecTrustGetCertificateCount

Returns the number of certificates in an evaluated certificate chain.

```
CFIndex SecTrustGetCertificateCount (
   SecTrustRef trust
);
```

Parameters

trust

The trust management object for the certificate that has been evaluated. Use the SecTrustCreateWithCertificates (page 25) function to create a trust management object and the SecTrustEvaluate (page 26) function to evaluate the certificate chain.

Return Value
The number of certificates in the certificate chain.

Discussion
You must call the SecTrustEvaluate (page 26) function before calling this function.

Availability
Available in iOS 2.0 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 34) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a SecTrustRef (page 34) 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 iOS 2.0 and later.

Declared In
SecTrust.h

SecTrustGetVerifyTime

Gets the absolute time against which the certificates in a trust management object are verified.

```
CFAbsoluteTime SecTrustGetVerifyTime (
   SecTrustRef trust
);
```

Parameters

trust

The trust management object whose verification time you want to get. A trust management object includes one or more certificates plus the policy or policies to be used in evaluating trust. Use the `SecTrustCreateWithCertificates` (page 25) function to create a trust management object.

Return Value

The absolute time at which the certificates should be checked for validity.

Discussion

This function returns the absolute time returned by:

1. the `CFDateGetAbsoluteTime` function for the date passed in to the `SecTrustSetVerifyDate` (page 32) function, if that was called, or
2. the last value returned by the `SecTrustGetVerifyTime` function, if it was called before, or
3. the value returned by the `CFAbsoluteTimeGetCurrent` function if neither `SecTrustSetVerifyDate` nor `SecTrustGetVerifyTime` were ever called.

It is safe to call this function concurrently on two or more threads as long as it is not used to get a value from a trust management object that is simultaneously being changed by another function. For example, you can call this function on two threads at the same time, but not if you are simultaneously calling the `SecTrustSetVerifyDate` (page 32) function for the same trust management object on another thread.

Availability

Available in iOS 2.0 and later.

See Also

`SecTrustSetVerifyDate` (page 32)

Declared In

`SecTrust.h`

SecTrustSetAnchorCertificates

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

```
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 25) 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 26) 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 39).

Discussion

The `SecTrustEvaluate` (page 26) 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 was 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` 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` function.

Note that when you call the `SecTrustSetAnchorCertificates` function, you are effectively telling the `SecTrustEvaluate` (page 26) function to use the anchor certificates in the specified array to evaluate trust regardless of any user-specified trust settings for those certificates. Furthermore, any intermediate certificates based on those anchor certificates are also trusted without consulting user trust settings.

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

It is safe to call this function concurrently on two or more threads as long as it is not used to change the value of a trust management object that is simultaneously being used by another function. For example, you cannot call this function on one thread at the same time as you are calling the `SecTrustEvaluate` (page 26) function for the same trust management object on another thread, but you can call this function and simultaneously evaluate a different trust management object on another thread. Similarly, calls to functions that return information about a trust management object (such as the `SecTrustCopyCustomAnchorCertificates` function) may fail or return an unexpected result if this function is simultaneously changing the same trust management object on another thread.

Important: Calling this function without also calling `SecTrustSetAnchorCertificatesOnly` (page 31) disables the trusting of any anchors other than the ones specified by this function call.

Availability

Available in iOS 2.0 and later.

See Also
SecTrustSetAnchorCertificatesOnly (page 31)

Declared In
SecTrust.h

SecTrustSetAnchorCertificatesOnly

Reenables trusting built-in anchor certificates.

```
OSStatus SecTrustSetAnchorCertificatesOnly (
   SecTrustRef trust,
   Boolean anchorCertificatesOnly
);
```

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 25) function to create a trust management object.

anchorCertificatesOnly

If true, disables trusting any anchors other than the ones passed in with the SecTrustSetAnchorCertificates (page 29) function. If false, the built-in anchor certificates are also trusted. If SecTrustSetAnchorCertificates is called and SecTrustSetAnchorCertificatesOnly is not called, only the anchors explicitly passed in are trusted.

Return Value
A result code. See "Certificate, Key, and Trust Services Result Codes" (page 39).

Discussion
It is safe to call this function concurrently on two or more threads as long as it is not used to change the value of a trust management object that is simultaneously being used by another function. For example, you cannot call this function on one thread at the same time as you are calling the SecTrustEvaluate (page 26) function for the same trust management object on another thread, but you can call this function and simultaneously evaluate a different trust management object on another thread. Similarly, calls to functions that return information about a trust management object (such as the SecTrustCopyCustomAnchorCertificates function) may fail or return an unexpected result if this function is simultaneously changing the same trust management object on another thread.

Availability
Available in iOS 2.0 and later.

See Also
SecTrustSetAnchorCertificates (page 29)

Declared In
SecTrust.h

SecTrustSetExceptions

Sets a list of exceptions that should be ignored when evaluating the certificate.

```
bool SecTrustSetExceptions(
    SecTrustRef trust,
    CFDataRef exceptions
);
```

Parameters

trust

The trust management object whose exception list you wish to modify.

exceptions

An opaque cookie returned by a prior call to `SecTrustCopyExceptions` (page 23).

Return Value

Returns `true` if the exceptions cookies was valid and matches the current leaf certificate, `false` otherwise.

Important: Even if this function returns true, you must still call `SecTrustEvaluate` (page 26) because the evaluation can still fail if something changes between the initial evaluation and the reevaluation.

Availability

Available in iOS 4.0 and later.

Declared In

`SecTrust.h`

SecTrustSetVerifyDate

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

```
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 25) 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 39).

Discussion

By default, the `SecTrustEvaluate` (page 26) 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.

It is safe to call this function concurrently on two or more threads as long as it is not used to change the value of a trust management object that is simultaneously being used by another function. For example, you cannot call this function on one thread at the same time as you are calling the `SecTrustEvaluate` (page 26) function for the same trust management object on another thread, but you can call this function and simultaneously evaluate a different trust management object on another thread. Similarly, calls to functions

that return information about a trust management object (such as the `SecTrustCopyCustomAnchorCertificates` function) may fail or return an unexpected result if this function is simultaneously changing the same trust management object on another thread.

Availability
Available in iOS 2.0 and later.

See Also
`SecTrustGetVerifyTime` (page 29)

Declared In
`SecTrust.h`

Data Types

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 iOS 2.0 and later.

Declared In
`SecBase.h`

SecIdentityRef

Abstract Core Foundation-type object representing an identity.

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

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

Availability
Available in iOS 2.0 and later.

Declared In
`SecBase.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 iOS 2.0 and later.

Declared In
`SecBase.h`

SecPolicyRef

Contains information about a policy.

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

Availability
Available in iOS 2.0 and later.

Declared In
`SecPolicy.h`

SecTrustRef

Contains information about trust management.

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

Availability
Available in iOS 2.0 and later.

Declared In
`SecTrust.h`

Constants

Digital Signature Padding Types

Specifies the type of padding to be used when creating or verifying a digital signature.

```
typedef uint32_t SecPadding;
enum
{
   kSecPaddingNone      = 0,
```

```
   kSecPaddingPKCS1     = 1,
   kSecPaddingPKCS1MD2  = 0x8000,
   kSecPaddingPKCS1MD5  = 0x8001,
   kSecPaddingPKCS1SHA1 = 0x8002,
};
```

Constants

`kSecPaddingNone`

No padding.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

`kSecPaddingPKCS1`

PKCS1 padding.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

`kSecPaddingPKCS1MD2`

Data to be signed is an MD2 hash.

Standard ASN.1 padding is done, as well as PKCS1 padding of the underlying RSA operation. Used with `SecKeyRawSign` (page 20) and `SecKeyRawVerify` (page 21) only.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

`kSecPaddingPKCS1MD5`

Data to be signed is an MD5 hash.

Standard ASN.1 padding is done, as well as PKCS1 padding of the underlying RSA operation. Used with `SecKeyRawSign` (page 20) and `SecKeyRawVerify` (page 21) only.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

`kSecPaddingPKCS1SHA1`

Data to be signed is a SHA1 hash.

Standard ASN.1 padding will be done, as well as PKCS1 padding of the underlying RSA operation. Used with `SecKeyRawSign` (page 20) and `SecKeyRawVerify` (page 21) only.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

Dictionary Key Constants For Key Generation

Use these dictionary keys with the `SecKeyGeneratePair` (page 17) function.

```
CFTypeRef kSecPrivateKeyAttrs;
CFTypeRef kSecPublicKeyAttrs;
```

Constants

`kSecPrivateKeyAttrs`

Private cryptographic key attributes.

The corresponding value is a `CFDictionaryRef` dictionary conatining key-value pairs for attributes specific to the private key to be generated.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

`kSecPublicKeyAttrs`

Public cryptographic key attributes.

The corresponding value is a `CFDictionaryRef` dictionary conatining key-value pairs for attributes specific to the public key to be generated.

Available in iOS 2.0 and later.

Declared in `SecKey.h`.

Trust Result Type Constants

Specifies the trust result type.

```
typedef uint32_t SecTrustResultType;
enum {
    kSecTrustResultInvalid,
    kSecTrustResultProceed,
    kSecTrustResultConfirm,
    kSecTrustResultDeny,
    kSecTrustResultUnspecified,
    kSecTrustResultRecoverableTrustFailure,
    kSecTrustResultFatalTrustFailure,
    kSecTrustResultOtherError
};
```

Constants

`kSecTrustResultInvalid`

Invalid setting or result. Usually, this result indicates that the `SecTrustEvaluate` (page 26) function did not complete successfully.

Available in iOS 2.0 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 26) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed “Always Trust.”

Available in iOS 2.0 and later.

Declared in `SecTrust.h`.

`kSecTrustResultConfirm`
: Confirmation from the user is required before proceeding. This value may be returned by the `SecTrustEvaluate` (page 26) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed "Ask Permission."

  Available in iOS 2.0 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 26) function or stored as part of the user trust settings. In the Keychain Access utility, this value is termed "Never Trust."

  Available in iOS 2.0 and later.

  Declared in `SecTrust.h`.

`kSecTrustResultUnspecified`
: The user did not specify a trust setting. This value may be returned by the `SecTrustEvaluate` (page 26) 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 iOS 2.0 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 `SecTrustSettingsSetTrustSettings` function to set the user trust setting to `kSecTrustResultProceed` and call `SecTrustEvaluate` (page 26) again. This value may be returned by the `SecTrustEvaluate` (page 26) function but not stored as part of the user trust settings.

  Available in iOS 2.0 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 26) function but not stored as part of the user trust settings.

  Available in iOS 2.0 and later.

  Declared in `SecTrust.h`.

`kSecTrustResultOtherError`
: A failure other than that of trust evaluation; for example, an internal failure of the `SecTrustEvaluate` (page 26) function. This value may be returned by the `SecTrustEvaluate` (page 26) function but not stored as part of the user trust settings.

  Available in iOS 2.0 and later.

  Declared in `SecTrust.h`.

Discussion

These constants may be returned by the `SecTrustEvaluate` (page 26) function or stored as one of the user trust settings (see `SecTrustSettingsSetTrustSettings`), as noted. When evaluating user trust, `SecTrustEvaluate` starts with the leaf certificate and works through the chain down to the anchor. The `SecTrustSettingsCopyTrustSettings` 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.

PKCS #12 Import Item Keys

Dictionary keys used in the dictionaries returned by the `SecPKCS12Import` (page 21) function.

```
extern CFStringRef kSecImportItemLabel;
extern CFStringRef kSecImportItemKeyID;
extern CFStringRef kSecImportItemTrust;
extern CFStringRef kSecImportItemCertChain;
extern CFStringRef kSecImportItemIdentity;
```

Constants

`kSecImportItemLabel`

Item label.

The corresponding value is of type `CFStringRef`. The format of the string is implementation specific.

Available in iOS 2.0 and later.

Declared in `SecImportExport.h`.

`kSecImportItemKeyID`

Key ID.

The corresponding value is of type `CFDataRef`. This unique ID is often the SHA-1 digest of the public encryption key.

Available in iOS 2.0 and later.

Declared in `SecImportExport.h`.

`kSecImportItemTrust`

Trust management object.

The corresponding value is of type `SecTrustRef`. The trust reference returned by the `SecPKCS12Import` (page 21) function has been evaluated against the basic X.509 policy and includes as complete a certificate chain as could be constructed from the certificates in the PKCS #12 blob, certificates on the keychain, and any other certificates available to the system. You can use the `SecTrustEvaluate` (page 26) function if you want to know whether the certificate chain is complete and valid (according to the basic X.509 policy). There is no guarantee that the evaluation will succeed.

Available in iOS 2.0 and later.

Declared in `SecImportExport.h`.

`kSecImportItemCertChain`

Certificate list.

The corresponding value is of type `CFArrayRef`. The array consists of `SecCertificateRef` objects for all the certificates in the PKCS #12 blob. This list might differ from that in the trust management object if there is more than one identity in the blob or if the blob contains extra certificates (for example, an intermediate certificate that is not yet valid but might be needed to establish validity in the near future).

Available in iOS 2.0 and later.

Declared in `SecImportExport.h`.

`kSecImportItemIdentity`

Identity object.

The corresponding value is of type `SecIdentityRef` and represents one identity contained in the PKCS #12 blob.

Available in iOS 2.0 and later.

Declared in `SecImportExport.h`.

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
`errSecSuccess`	0	No error. Available in iOS 2.0 and later.
`errSecUnimplemented`	-4	The function or operation is not implemented. Available in iOS 2.0 and later.
`errSecParam`	-50	One or more parameters passed to a function were not valid. Available in iOS 2.0 and later.
`errSecAllocate`	-108	Failed to allocate memory. Available in iOS 2.0 and later.
`errSecNotAvailable`	–25291	No keychain is available. Available in iOS 2.0 and later.
`errSecAuthFailed`	–25293	Authorization or authentication failed. Available in iOS 4.2 and later.
`errSecDuplicateItem`	–25299	An item with the same primary key attributes already exists. Available in iOS 2.0 and later.
`errSecItemNotFound`	–25300	The item cannot be found. Available in iOS 2.0 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 iOS 2.0 and later.
`errSecDecode`	-26275	Unable to decode the provided data. Available in iOS 2.0 and later.

Keychain Services Reference

Framework:	Security/Security.h
Declared in	SecItem.h SecAccess.h SecACL.h SecBase.h SecImportExport.h SecKeychain.h SecKeychainItem.h SecKeychainSearch.h SecTrustedApplication.h

Overview

Keychain Services is a programming interface that enables you to find, add, modify, and delete keychain items.

Functions by Task

Using Keychain Item Search Dictionaries

For this interface, keychain items are found or defined by a CFDictionary of key-value pairs. Each key in the dictionary identifies one attribute of the keychain item, or a search option. For example, you can use the `kSecClass` key to specify that the keychain item is an Internet password, that it has a specific creation date, that it is for the HTTPS protocol, and that only the first match found should be returned. The keys that can be used for this purpose and the possible values for each key are listed in the "Keychain Services Constants" (page 47) section.

See the discussion section of the `SecItemCopyMatching` (page 43) function for information about how to construct a keychain-item search dictionary.

`SecItemCopyMatching` (page 43)
: Returns one or more keychain items that match a search query.

`SecItemAdd` (page 42)
: Adds one or more items to a keychain.

`SecItemUpdate` (page 46)
: Modifies items that match a search query.

`SecItemDelete` (page 45)
Deletes items that match a search query.

Functions

SecItemAdd

Adds one or more items to a keychain.

```
OSStatus SecItemAdd (
   CFDictionaryRef attributes,
   CFTypeRef *result
);
```

Parameters

attributes
A dictionary containing an item class key-value pair ("Keychain Item Class Keys and Values" (page 47)) and optional attribute key-value pairs ("Attribute Item Keys and Values" (page 50)) specifying the item's attribute values.

result
On return, a reference to the newly added items. The exact type of the result is based on the values supplied in attributes, as discussed below. Pass `NULL` if this result is not required.

Return Value

A result code. See "Keychain Services Result Codes" (page 69). Call `SecCopyErrorMessageString` (Mac OS X only) to get a human-readable string explaining the result.

Discussion

You specify attributes defining an item by adding key-value pairs to the attributes dictionary. To add multiple items to a keychain at once use the `kSecUseItemList` key (see section "Item List Key" (page 67)) with an array of items as its value. This is currently only supported for non-password items.

If you want the new keychain item to be shared among multiple applications, include the `kSecAttrAccessGroup` (page 57) key in the attributes dictionary. The value of this key must be the name of a keychain access group to which all of the programs that will share this item belong.

When you use Xcode to create an application, Xcode adds an application-identifier entitlement to the application bundle. Keychain Services uses this entitlement to grant the application access to its own keychain items. You can also add a keychain-access-groups entitlement to the application and, in the entitlement property list file, specify an array of keychain access groups to which the application belongs. The property list file can have any name you like (for example, `keychain-access-groups.plist`). The Xcode build variable `CODE_SIGN_ENTITLEMENTS` should contain the `SRCROOT` relative path to the entitlement property list file. The property list file itself should be a dictionary with a top-level key called `keychain-access-groups` whose value is an array of strings. If you add such a property-list file to the application bundle, then the access group corresponding to the application-identifier entitlement is treated as the last element in the access groups array. If you do not include the `kSecAttrAccessGroup` (page 57) key in the attributes dictionary when you call the `SecItemAdd` function to add an item to the keychain, the function uses the first access group in the array by default. If there is no `kSecAttrAccessGroup` key in the attributes dictionary and there is no keychain-access-groups entitlement in the application bundle, then the access group of a newly created item is the value of the application-identifier entitlement.

For example, a development group in Apple might have the ID:

`659823F3DC53.com.apple`

and the application identifiers of their two applications might be:

`659823F3DC53.com.apple.oneappleapp` and

`659823F3DC53.com.apple.twoappleapp`

If both applications add a keychain-access-groups entitlement with one value in the array of access groups:

`659823F3DC53.com.apple.netaccount`

then both applications would add new keychain items to the `659823F3DC53.com.apple.netaccount` access group by default and both applications would have access to keychain items in that group. In addition, each application would still have access to its own private keychain items: OneAppleApp would have access to items in keychain access group `659823F3DC53.com.apple.oneappleapp` and TwoAppleApp would have access to items in `659823F3DC53.com.apple.twoappleapp`.

Return types ("Search Results Constants" (page 67)) are specified as follows:

- To obtain the data of the added item as an object of type `CFDataRef`, specify the return type key `kSecReturnData` with a value of `kCFBooleanTrue`.
- To obtain all the attributes of the added item as objects of type `CFDictionaryRef`, specify `kSecReturnAttributes` with a value of `kCFBooleanTrue`.
- To obtain a reference to the added item of type `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, or `SecIdentityRef`), specify `kSecReturnRef` with a value of `kCFBooleanTrue`. This is the default behavior if a return type is not explicitly specified.
- To obtain a persistent reference to the added item (an object of type `CFDataRef`), specify `kSecReturnPersistentRef` with a value of `kCFBooleanTrue`. Note that unlike normal references, a persistent reference may be stored on disk or passed between processes.
- If more than one of these return types is specified, the result is returned as an object of type `CFDictionaryRef` containing all the requested data.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise
GenericKeychain

Declared In
`SecItem.h`

SecItemCopyMatching

Returns one or more keychain items that match a search query.

```
OSStatus SecItemCopyMatching (
   CFDictionaryRef query,
   CFTypeRef *result
);
```

Parameters

query

A dictionary containing an item class specification ("Keychain Item Class Keys and Values" (page 47)) and optional attributes for controlling the search. See "Keychain Services Constants" (page 47) for a description of currently defined search attributes.

result

On return, a reference to the found items. The exact type of the result is based on the search attributes supplied in the query, as discussed below.

Return Value

A result code. See "Keychain Services Result Codes" (page 69). Call `SecCopyErrorMessageString` (Mac OS X only) to get a human-readable string explaining the result.

Discussion

You specify attributes defining a search by adding key-value pairs to the query dictionary.

A typical query consists of:

- The class key ("Item Class Key Constant" (page 47)) and a class value constant ("Item Class Value Constants" (page 47)), which specifies the class of items for which to search.
- One or more attribute key-value pairs ("Attribute Item Keys and Values" (page 50)), which specify the attribute data to be matched.
- One or more search key-value pairs ("Search Keys" (page 65)), which specify values that further refine the search.
- A return-type key-value pair ("Search Results Constants" (page 67)), specifying the type of results you desire.

Return types ("Search Results Constants" (page 67)) are specified as follows:

- To obtain a reference (of type `CFDataRef`) to the data of a matching item, specify `kSecReturnData` with a value of `kCFBooleanTrue`.
- To obtain a dictionary (of type `CFDictionaryRef`) containing the attributes of a matching item, specify `kSecReturnAttributes` with a value of `kCFBooleanTrue`.
- To obtain a reference (of type `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, or `SecIdentityRef`) to a matching item, specify `kSecReturnRef` with a value of `kCFBooleanTrue`.
- To obtain a persistent reference (of type `CFDataRef`) to a matching item, specify `kSecReturnPersistentRef` with a value of `kCFBooleanTrue`. Note that unlike normal references, a persistent reference may be stored on disk or passed between processes.
- If more than one of these return types is specified, the result is returned as a dictionary (that is, an object of type `CFDictionaryRef`) containing all the requested data.

By default, this function returns only the first match found. To obtain more than one matching item at a time, specify the search key `kSecMatchLimit` with a value greater than `1`. The result will be an object of type `CFArrayRef` containing up to that number of matching items.

By default, this function searches for items in the keychain. To instead provide your own set of items to be filtered by this search query, specify the search key `kSecMatchItemList` with a value that consists of an object of type `CFArrayRef` referencing an array that contains items of type either `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, or `SecIdentityRef`. The objects in the provided array must all be of the same type.

To convert from persistent item references to normal item references, specify the search key `kSecMatchItemList` with a value that consists of an object of type `CFArrayRef` referencing an array containing one or more elements of type `CFDataRef` (the persistent references), and a return-type key of `kSecReturnRef` whose value is `kCFBooleanTrue`. The objects in the provided array must all be of the same type.

When you use Xcode to create an application, Xcode adds an application-identifier entitlement to the application bundle. Keychain Services uses this entitlement to grant the application access to its own keychain items. You can also add a keychain-access-groups entitlement to the application and, in the entitlement property list file, specify an array of keychain access groups to which the application belongs. The property list file can have any name you like (for example, `keychain-access-groups.plist`). The Xcode build variable `CODE_SIGN_ENTITLEMENTS` should contain the `SRCROOT` relative path to the entitlement property list file. The property list file itself should be a dictionary with a top-level key called `keychain-access-groups` whose value is an array of strings. When you call the `SecItemAdd` (page 42) function to add an item to the keychain, you can specify the access group to which that item should belong. By default, the `SecItemCopyMatching` function searches all the access groups to which the application belongs. However, you can add the `kSecAttrAccessGroup` (page 57) key to the search dictionary to specify which access group to search for keychain items.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise
GenericKeychain

Declared In
`SecItem.h`

SecItemDelete

Deletes items that match a search query.

```
OSStatus SecItemDelete (
   CFDictionaryRef query
);
```

Parameters

query

A dictionary containing an item class specification and optional attributes for controlling the search. See "Search Keys" (page 65) for a description of currently defined search attributes.

Return Value
A result code. See "Keychain Services Result Codes" (page 69). Call `SecCopyErrorMessageString` (Mac OS X only) to get a human-readable string explaining the result.

Discussion
See the discussion section of the `SecItemCopyMatching` (page 43) function for information about how to construct a search dictionary.

By default, this function deletes all items matching the specified query. You can change this behavior by specifying a key, as follows:

- To delete an item identified by a transient reference, specify the `kSecMatchItemList` search key with a reference returned by using the `kSecReturnRef` return type key in a previous call to the `SecItemCopyMatching` (page 43) or `SecItemAdd` (page 42) functions.
- To delete an item identified by a persistent reference, specify the `kSecMatchItemList` search key with a persistent reference returned by using the `kSecReturnPersistentRef` return type key to the `SecItemCopyMatching` (page 43) or `SecItemAdd` (page 42) functions.
- If more than one of these return keys is specified, the behavior is undefined.

Availability
Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise
GenericKeychain

Declared In
`SecItem.h`

SecItemUpdate

Modifies items that match a search query.

```
OSStatus SecItemUpdate (
   CFDictionaryRef query,
   CFDictionaryRef attributesToUpdate
);
```

Parameters

query
A dictionary containing an item class specification and optional attributes for controlling the search. Specify the items whose values you wish to change. See "Search Keys" (page 65) for a description of currently defined search attributes.

attributesToUpdate
A dictionary containing the attributes whose values should be changed, along with the new values. Only real keychain attributes are permitted in this dictionary (no "meta" attributes are allowed.) See "Attribute Item Keys and Values" (page 50) for a description of currently defined value attributes.

Return Value
A result code. See "Keychain Services Result Codes" (page 69). Call `SecCopyErrorMessageString` (Mac OS X only) to get a human-readable string explaining the result.

Discussion
See the discussion section of the `SecItemCopyMatching` (page 43) function for information about how to construct a search dictionary.

Availability
Available in iOS 2.0 and later.

Related Sample Code
GenericKeychain

Declared In
`SecItem.h`

Constants

Keychain Item Class Keys and Values

Constants used in a search dictionary to specify the class of items in the keychain. See `SecItemCopyMatching` (page 43) for a description of a search dictionary.

Item Class Key Constant

Key constant used to set the item class value in a search dictionary.

```
CFTypeRef kSecClass;
```

Constants
`kSecClass`
Dictionary key whose value is the item's class code.
Possible values for this key are listed in "Item Class Value Constants" (page 47).
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

Item Class Value Constants

Values used with the `kSecClass` key in a search dictionary.

```
CFTypeRef kSecClassGenericPassword;
CFTypeRef kSecClassInternetPassword;
CFTypeRef kSecClassCertificate;
CFTypeRef kSecClassKey;
CFTypeRef kSecClassIdentity;
```

Constants

`kSecClassGenericPassword`

Generic password item.

The following attribute types ("Attribute Item Keys and Values" (page 50)) can be used with an item of this type:

`kSecAttrAccessible`
`kSecAttrAccessGroup`
`kSecAttrCreationDate`
`kSecAttrModificationDate`
`kSecAttrDescription`
`kSecAttrComment`
`kSecAttrCreator`
`kSecAttrType`
`kSecAttrLabel`
`kSecAttrIsInvisible`
`kSecAttrIsNegative`
`kSecAttrAccount`
`kSecAttrService`
`kSecAttrGeneric`

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecClassInternetPassword`

Internet password item.

The following attribute types ("Attribute Item Keys and Values" (page 50)) can be used with an item of this type:

`kSecAttrAccessible`
`kSecAttrAccessGroup`
`kSecAttrCreationDate`
`kSecAttrModificationDate`
`kSecAttrDescription`
`kSecAttrComment`
`kSecAttrCreator`
`kSecAttrType`
`kSecAttrLabel`
`kSecAttrIsInvisible`
`kSecAttrIsNegative`
`kSecAttrAccount`
`kSecAttrSecurityDomain`
`kSecAttrServer`
`kSecAttrProtocol`
`kSecAttrAuthenticationType`
`kSecAttrPort`
`kSecAttrPath`

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecClassCertificate`

Certificate item.

The following attribute types ("Attribute Item Keys and Values" (page 50)) can be used with an item of this type:

`kSecAttrAccessible`
`kSecAttrAccessGroup`
`kSecAttrCertificateType`
`kSecAttrCertificateEncoding`
`kSecAttrLabel`
`kSecAttrSubject`
`kSecAttrIssuer`
`kSecAttrSerialNumber`
`kSecAttrSubjectKeyID`
`kSecAttrPublicKeyHash`

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecClassKey`

Cryptographic key item.

The following attribute types ("Attribute Item Keys and Values" (page 50)) can be used with an item of this type:

`kSecAttrAccessible`
`kSecAttrAccessGroup`
`kSecAttrKeyClass`
`kSecAttrLabel`
`kSecAttrApplicationLabel`
`kSecAttrIsPermanent`
`kSecAttrApplicationTag`
`kSecAttrKeyType`
`kSecAttrKeySizeInBits`
`kSecAttrEffectiveKeySize`
`kSecAttrCanEncrypt`
`kSecAttrCanDecrypt`
`kSecAttrCanDerive`
`kSecAttrCanSign`
`kSecAttrCanVerify`
`kSecAttrCanWrap`
`kSecAttrCanUnwrap`

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecClassIdentity`

Identity item.

An identity is a certificate together with its associated private key. Because an identity is the combination of a private key and a certificate, this class shares attributes of both `kSecClassKey` and `kSecClassCertificate`.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

Attribute Item Keys and Values

You use keys in a search dictionary to specify the keychain items for which to search. You can specify a combination of item attributes and search attributes (see "Search Keys" (page 65)) when looking for matching items with the `SecItemCopyMatching` (page 43) function. This section lists all the keys that specify keychain item attributes. The description of each item indicates what the possible values are for that key. In a few cases, the programming interface provides a set of constants that you can use as values for a specific key. Those value constants are also in this section, following the descriptions of the keys.

Attribute Item Keys

Each type of keychain item can have a number of attributes describing that item. For the possible types of keychain item and the attributes that can be specified for each, see "Keychain Item Class Keys and Values" (page 47).

```
CFTypeRef kSecAttrAccessible;
CFTypeRef kSecAttrCreationDate;
CFTypeRef kSecAttrModificationDate;
CFTypeRef kSecAttrDescription;
CFTypeRef kSecAttrComment;
CFTypeRef kSecAttrCreator;
CFTypeRef kSecAttrType;
CFTypeRef kSecAttrLabel;
CFTypeRef kSecAttrIsInvisible;
CFTypeRef kSecAttrIsNegative;
CFTypeRef kSecAttrAccount;
CFTypeRef kSecAttrService;
CFTypeRef kSecAttrGeneric;
CFTypeRef kSecAttrSecurityDomain;
CFTypeRef kSecAttrServer;
CFTypeRef kSecAttrProtocol;
CFTypeRef kSecAttrAuthenticationType;
CFTypeRef kSecAttrPort;
CFTypeRef kSecAttrPath;
CFTypeRef kSecAttrSubject;
CFTypeRef kSecAttrIssuer;
CFTypeRef kSecAttrSerialNumber;
CFTypeRef kSecAttrSubjectKeyID;
CFTypeRef kSecAttrPublicKeyHash;
CFTypeRef kSecAttrCertificateType;
CFTypeRef kSecAttrCertificateEncoding;
CFTypeRef kSecAttrKeyClass;
CFTypeRef kSecAttrApplicationLabel;
CFTypeRef kSecAttrIsPermanent;
CFTypeRef kSecAttrApplicationTag;
CFTypeRef kSecAttrKeyType;
CFTypeRef kSecAttrKeySizeInBits;
CFTypeRef kSecAttrEffectiveKeySize;
CFTypeRef kSecAttrCanEncrypt;
CFTypeRef kSecAttrCanDecrypt;
CFTypeRef kSecAttrCanDerive;
CFTypeRef kSecAttrCanSign;
CFTypeRef kSecAttrCanVerify;
CFTypeRef kSecAttrCanWrap;
CFTypeRef kSecAttrCanUnwrap;
CFTypeRef kSecAttrAccessGroup;
```

Constants

`kSecAttrAccessible`

A `CFTypeRef` (opaque) value that indicates when your application needs access to the data in a keychain item. You should choose the most restrictive option that meets your application's needs so that iOS can protect that item to the greatest extent possible. For a list of possible values, see "Keychain Item Accessibility Constants" (page 63).

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

`kSecAttrCreationDate`
: Creation date key.

  The corresponding value is of type `CFDateRef` and represents the date the item was created. Read only.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrModificationDate`
: Modification date key.

  The corresponding value is of type `CFDateRef` and represents the last time the item was updated. Read only.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrDescription`
: Description attribute key.

  The corresponding value is of type `CFStringRef` and specifies a user-visible string describing this kind of item (for example, "Disk image password").

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrComment`
: Comment attribute key.

  The corresponding value is of type `CFStringRef` and contains the user-editable comment for this item.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCreator`
: Creator attribute key.

  The corresponding value is of type `CFNumberRef` and represents the item's creator. This number is the unsigned integer representation of a four-character code (for example, `'aCrt'`).

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrType`
: Type attribute key.

  The corresponding value is of type `CFNumberRef` and represents the item's type. This number is the unsigned integer representation of a four-character code (for example, 'aTyp').

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrLabel`
: Label attribute key.

  The corresponding value is of type `CFStringRef` and contains the user-visible label for this item.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrIsInvisible`
: Invisible attribute key.

  The corresponding value is of type `CFBooleanRef` and is `kCFBooleanTrue` if the item is invisible (that is, should not be displayed).

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrIsNegative`
: Negative attribute key.

  The corresponding value is of type `CFBooleanRef` and 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.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrAccount`
: Account attribute key.

  The corresponding value is of type `CFStringRef` and contains an account name. Items of class `kSecClassGenericPassword` and `kSecClassInternetPassword` have this attribute.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrService`
: Service attribute key.

  The corresponding value is a string of type `CFStringRef` that represents the service associated with this item. Items of class `kSecClassGenericPassword` have this attribute.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrGeneric`
: Generic attribute key.

  The corresponding value is of type `CFDataRef` and contains a user-defined attribute. Items of class `kSecClassGenericPassword` have this attribute.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrSecurityDomain`
: Security domain attribute key.

  The corresponding value is of type `CFStringRef` and represents the Internet security domain. Items of class `kSecClassInternetPassword` have this attribute.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrServer`
: Server attribute key.

  The corresponding value is of type `CFStringRef` and contains the server's domain name or IP address. Items of class `kSecClassInternetPassword` have this attribute.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrProtocol`
Protocol attribute key.

The corresponding value is of type `CFNumberRef` and denotes the protocol for this item (see "Protocol Values" (page 58)). Items of class `kSecClassInternetPassword` have this attribute.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrAuthenticationType`
Authentication type attribute key.

The corresponding value is of type `CFNumberRef` and denotes the authentication scheme for this item (see "Authentication Type Values" (page 61)).

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrPort`
Port attribute key.

The corresponding value is of type `CFNumberRef` and represents an Internet port number. Items of class `kSecClassInternetPassword` have this attribute.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrPath`
Path attribute key.

The corresponding value is of type `CFStringRef` and represents a path, typically the path component of the URL. Items of class `kSecClassInternetPassword` have this attribute.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrSubject`
Subject attribute key.

The corresponding value is of type `CFDataRef` and contains the X.500 subject name of a certificate. Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrIssuer`
Issuer attribute key.

The corresponding value is of type `CFDataRef` and contains the X.500 issuer name of a certificate. Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrSerialNumber`
Serial number attribute key.

The corresponding value is of type `CFDataRef` and contains the serial number data of a certificate. Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrSubjectKeyID`
Subject key ID attribute key.

The corresponding value is of type `CFDataRef` and contains the subject key ID of a certificate. Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrPublicKeyHash`
Public key hash attribute key.

The corresponding value is of type `CFDataRef` and contains the hash of a certificate's public key. Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrCertificateType`
Certificate type attribute key.

The corresponding value is of type `CFNumberRef` and denotes the certificate type (see the `CSSM_CERT_TYPE` enumeration in cssmtype.h). Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrCertificateEncoding`
Certificate encoding attribute key.

The corresponding value is of type `CFNumberRef` and denotes the certificate encoding (see the `CSSM_CERT_ENCODING` enumeration in cssmtype.h). Items of class `kSecClassCertificate` have this attribute. Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrKeyClass`
Key class attribute key.

The corresponding value is of type `CFTypeRef` and specifies a type of cryptographic key. Possible values are listed in "Key Class Values" (page 63). Read only.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrApplicationLabel`
Application label attribute key.

The corresponding value is of type `CFStringRef` and contains a label for this item. This attribute is different from the `kSecAttrLabel` attribute, which is intended to be human-readable. This attribute is used to look up a key programmatically; in particular, for keys of class `kSecAttrKeyClassPublic` and `kSecAttrKeyClassPrivate`, the value of this attribute is the hash of the public key.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrIsPermanent`
: Permanence attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key is to be stored permanently.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrApplicationTag`
: Private tag attribute key.

  The corresponding value is of type `CFDataRef` and contains private tag data.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrKeyType`
: Algorithm attribute key.

  The corresponding value is of type `CFNumberRef` and indicates the algorithm associated with this cryptographic key (see the `CSSM_ALGORITHMS` enumeration in cssmtype.h and "Key Type Value" (page 63)).

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrKeySizeInBits`
: Number of bits attribute key.

  The corresponding value is of type `CFNumberRef` and indicates the total number of bits in this cryptographic key. Compare with `kSecAttrEffectiveKeySize`.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrEffectiveKeySize`
: Effective number of bits attribute key.

  The corresponding value is of type `CFNumberRef` and indicates the effective number of bits in this cryptographic key. For example, a DES key has a `kSecAttrKeySizeInBits` of 64, but a `kSecAttrEffectiveKeySize` of 56 bits.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanEncrypt`
: Encryption attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to encrypt data.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanDecrypt`
: Decryption attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to decrypt data.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanDerive`
: Derivation attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to derive another key.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanSign`
: Signature attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to create a digital signature.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanVerify`
: Signature verification attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to verify a digital signature.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanWrap`
: Wrap attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to wrap another key.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrCanUnwrap`
: Unwrap attribute key.

  The corresponding value is of type `CFBooleanRef` and indicates whether this cryptographic key can be used to unwrap another key.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecAttrAccessGroup`
: Access group key.

  The corresponding value is of type `CFStringRef` and indicates which access group an item is in. Access groups can be used to share keychain items among two or more applications. For applications to share a keychain item, the applications must have a common access group listed in their keychain-access-groups entitlement, and the application adding the shared item to the keychain must specify this shared access-group name as the value for this key in the dictionary passed to the `SecItemAdd` (page 42) function.

  An application can be a member of any number of access groups. By default, the `SecItemUpdate` (page 46), `SecItemDelete` (page 45), and `SecItemCopyMatching` (page 43) functions search all the access groups an application is a member of. Include this key in the search dictionary for these functions to specify which access group is searched.

  A keychain item can be in only a single access group.

  Available in iOS 3.0 and later.

  Declared in `SecItem.h`.

Discussion
These predefined item attribute keys are used to get or set values in a dictionary. Not all attributes apply to each item class.

Protocol Values

Values that can be used with the `kSecAttrProtocol` attribute key.

```
CFTypeRef kSecAttrProtocolFTP;
CFTypeRef kSecAttrProtocolFTPAccount;
CFTypeRef kSecAttrProtocolHTTP;
CFTypeRef kSecAttrProtocolIRC;
CFTypeRef kSecAttrProtocolNNTP;
CFTypeRef kSecAttrProtocolPOP3;
CFTypeRef kSecAttrProtocolSMTP;
CFTypeRef kSecAttrProtocolSOCKS;
CFTypeRef kSecAttrProtocolIMAP;
CFTypeRef kSecAttrProtocolLDAP;
CFTypeRef kSecAttrProtocolAppleTalk;
CFTypeRef kSecAttrProtocolAFP;
CFTypeRef kSecAttrProtocolTelnet;
CFTypeRef kSecAttrProtocolSSH;
CFTypeRef kSecAttrProtocolFTPS;
CFTypeRef kSecAttrProtocolHTTPS;
CFTypeRef kSecAttrProtocolHTTPProxy;
CFTypeRef kSecAttrProtocolHTTPSProxy;
CFTypeRef kSecAttrProtocolFTPProxy;
CFTypeRef kSecAttrProtocolSMB;
CFTypeRef kSecAttrProtocolRTSP;
CFTypeRef kSecAttrProtocolRTSPProxy;
CFTypeRef kSecAttrProtocolDAAP;
CFTypeRef kSecAttrProtocolEPPC;
CFTypeRef kSecAttrProtocolIPP;
CFTypeRef kSecAttrProtocolNNTPS;
CFTypeRef kSecAttrProtocolLDAPS;
CFTypeRef kSecAttrProtocolTelnetS;
CFTypeRef kSecAttrProtocolIMAPS;
CFTypeRef kSecAttrProtocolIRCS;
CFTypeRef kSecAttrProtocolPOP3S;
```

Constants

`kSecAttrProtocolFTP`

FTP protocol.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolFTPAccount`

A client side FTP account.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolHTTP`
HTTP protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolIRC`
IRC protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolNNTP`
NNTP protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolPOP3`
POP3 protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolSMTP`
SMTP protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolSOCKS`
SOCKS protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolIMAP`
IMAP protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolLDAP`
LDAP protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolAppleTalk`
AFP over AppleTalk.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolAFP`
AFP over TCP.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolTelnet`
Telnet protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolSSH`
SSH protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolFTPS`
FTP over TLS/SSL.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolHTTPS`
HTTP over TLS/SSL.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolHTTPProxy`
HTTP proxy.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolHTTPSProxy`
HTTPS proxy.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolFTPProxy`
FTP proxy.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolSMB`
SMB protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolRTSP`
RTSP protocol.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolRTSPProxy`
RTSP proxy.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrProtocolDAAP`
DAAP protocol.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolEPPC`
Remote Apple Events.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolIPP`
IPP protocol.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolNNTPS`
NNTP over TLS/SSL.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolLDAPS`
LDAP over TLS/SSL.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolTelnetS`
Telnet over TLS/SSL.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolIMAPS`
IMAP over TLS/SSL.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolIRCS`
IRC over TLS/SSL.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecAttrProtocolPOP3S`
POP3 over TLS/SSL.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

Authentication Type Values

Values that can be used with the `kSecAttrAuthenticationType` attribute key.

```
CFTypeRef kSecAttrAuthenticationTypeNTLM;
CFTypeRef kSecAttrAuthenticationTypeMSN;
CFTypeRef kSecAttrAuthenticationTypeDPA;
CFTypeRef kSecAttrAuthenticationTypeRPA;
CFTypeRef kSecAttrAuthenticationTypeHTTPBasic;
CFTypeRef kSecAttrAuthenticationTypeHTTPDigest;
CFTypeRef kSecAttrAuthenticationTypeHTMLForm;
CFTypeRef kSecAttrAuthenticationTypeDefault;
```

Constants

`kSecAttrAuthenticationTypeNTLM`
Windows NT LAN Manager authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeMSN`
Microsoft Network default authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeDPA`
Distributed Password authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeRPA`
Remote Password authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeHTTPBasic`
HTTP Basic authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeHTTPDigest`
HTTP Digest Access authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeHTMLForm`
HTML form based authentication.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrAuthenticationTypeDefault`
The default authentication type.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

Key Class Values

Values that can be used with the `kSecAttrKeyClass` attribute key.

```
CFTypeRef kSecAttrKeyClassPublic;
CFTypeRef kSecAttrKeyClassPrivate;
CFTypeRef kSecAttrKeyClassSymmetric;
```

Constants

`kSecAttrKeyClassPublic`
A public key of a public-private pair.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrKeyClassPrivate`
A private key of a public-private pair.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrKeyClassSymmetric`
A private key used for symmetric-key encryption and decryption.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

Key Type Value

A values that can be used with the `kSecAttrKeyType` attribute key.

```
CFTypeRef kSecAttrKeyTypeRSA;
```

Constants

`kSecAttrKeyTypeRSA`
RSA algorithm.
Available in iOS 2.0 and later.
Declared in `SecItem.h`.

`kSecAttrKeyTypeEC`
Elliptic curve algorithm.
Available in iOS 4.0 and later.
Declared in `SecItem.h`.

Keychain Item Accessibility Constants

These constants are legal values for `kSecAttrAccessible` (page 51) used for determining when a keychain item should be readable.

```
CFTypeRef kSecAttrAccessibleWhenUnlocked;
CFTypeRef kSecAttrAccessibleAfterFirstUnlock;
CFTypeRef kSecAttrAccessibleAlways;
CFTypeRef kSecAttrAccessibleWhenUnlockedThisDeviceOnly;
CFTypeRef kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly;
CFTypeRef kSecAttrAccessibleAlwaysThisDeviceOnly;
```

Constants

`kSecAttrAccessibleAfterFirstUnlock`

The data in the keychain item cannot be accessed after after a restart until the device has been unlocked once by the user. After the first unlock, the data remains accessible until the next restart. This is recommended for items that need to be accessed by background applications. Items with this attribute migrate to a new device when using encrypted backups.

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

`kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly`

The data in the keychain item cannot be accessed after after a restart until the device has been unlocked once by the user. After the first unlock, the data remains accessible until the next restart. This is recommended for items that need to be accessed by background applications. Items with this attribute *do not* migrate to a new device or new installation. Thus, after restoring from a backup, these items will not be present.

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

`kSecAttrAccessibleAlways`

The data in the keychain item can always be accessed regardless of whether the device is locked. This is not recommended for application use. Items with this attribute migrate to a new device when using encrypted backups.

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

`kSecAttrAccessibleAlwaysThisDeviceOnly`

The data in the keychain item can always be accessed regardless of whether the device is locked. This is not recommended for application use. Items with this attribute *do not* migrate to a new device or new installation. Thus, after restoring from a backup, these items will not be present.

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

`kSecAttrAccessibleWhenUnlocked`

The data in the keychain item can be accessed only while the device is unlocked by the user. This is recommended for items that need to be accesible only while the application is in the foreground. Items with this attribute migrate to a new device when using encrypted backups.

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

`kSecAttrAccessibleWhenUnlockedThisDeviceOnly`

The data in the keychain item can be accessed only while the device is unlocked by the user. This is recommended for items that need to be accesible only while the application is in the foreground. Items with this attribute *do not* migrate to a new device or new installation. Thus, after restoring from a backup, these items will not be present.

Available in iOS 4.0 and later.

Declared in `SecItem.h`.

Search Keys

Search Attribute Keys

Keys used to set search attributes in a keychain search dictionary. You can specify a combination of search attributes and item attributes (see "Attribute Item Keys and Values" (page 50)) when looking for matching items with the `SecItemCopyMatching` (page 43) function.

```
CFTypeRef kSecMatchPolicy;
CFTypeRef kSecMatchItemList;
CFTypeRef kSecMatchSearchList;
CFTypeRef kSecMatchIssuers;
CFTypeRef kSecMatchEmailAddressIfPresent;
CFTypeRef kSecMatchSubjectContains;
CFTypeRef kSecMatchCaseInsensitive;
CFTypeRef kSecMatchTrustedOnly;
CFTypeRef kSecMatchValidOnDate;
CFTypeRef kSecMatchLimit;
CFTypeRef kSecMatchLimitOne;
CFTypeRef kSecMatchLimitAll;
```

Constants

`kSecMatchPolicy`

Match policy attribute key.

The corresponding value is of type `SecPolicyRef`. If provided, returned certificates or identities must verify with this policy.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchItemList`

Item list attribute key.

To provide your own set of items to be filtered by a search query rather than searching the keychain, specify this search key in a call to the `SecItemCopyMatching` (page 43) function with a value that consists of an object of type `CFArrayRef` where the array contains either `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, `SecIdentityRef`, or `CFDataRef` items. The objects in the provided array must all be of the same type.

To convert from persistent item references to normal item references, specify this search key in a call to the `SecItemCopyMatching` (page 43) function with a value of type `CFArrayRef` where the array contains one or more `CFDataRef` elements (the persistent references), and a return-type key of `kSecReturnRef` whose value is `kCFBooleanTrue`.

To delete an item identified by a transient reference, specify the `kSecMatchItemList` search key in a call to the `SecItemDelete` (page 45) function with a reference returned by using the `kSecReturnRef` return type key in a previous call to the `SecItemCopyMatching` (page 43) or `SecItemAdd` (page 42) functions.

To delete an item identified by a persistent reference, specify the `kSecMatchItemList` search key in a call to the `SecItemDelete` (page 45) function with a persistent reference returned by using the `kSecReturnPersistentRef` return type key to the `SecItemCopyMatching` (page 43) or `SecItemAdd` (page 42) functions.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchSearchList`

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchIssuers`

Issuers attribute key.

The corresponding value is of type `CFArrayRef`, where the array consists of X.500 names of type `CFDataRef`. If provided, returned certificates or identities are limited to those whose certificate chain contains one of the issuers provided in this list.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchEmailAddressIfPresent`

Email address attribute key.

The corresponding value is of type `CFStringRef` and contains an RFC822 email address. If provided, returned certificates or identities are limited to those that either contain the address or do not contain any email address.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchSubjectContains`

Subject attribute key.

The corresponding value is of type `CFStringRef`. If provided, returned certificates or identities are limited to those whose subject contains this string.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchCaseInsensitive`

Case sensitivity attribute key.

The corresponding value is of type `CFBooleanRef`. If this value is `kCFBooleanFalse`, or if this attribute is not provided, then case-sensitive string matching is performed.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchTrustedOnly`

Trusted anchor attribute key.

The corresponding value is of type `CFBooleanRef`. If this attribute is provided with A value of `kCFBooleanTrue`, only certificates that can be verified back to a trusted anchor are returned. If this value is `kCFBooleanFalse` or the attribute is not provided, then both trusted and untrusted certificates may be returned.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchValidOnDate`

Valid-on-date attribute key.

The corresponding value is of type `CFDateRef`. If provided, returned keys, certificates or identities are limited to those that are valid for the given date. Pass a value of `kCFNull` to indicate the current date.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecMatchLimit`
: Match limit attribute key.

  The corresponding value is of type `CFNumberRef`. If provided, this value specifies the maximum number of results to return. If not provided, results are limited to the first item found. For a single item, specify `kSecMatchLimitOne`. To return all matching items, specify `kSecMatchLimitAll`.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecMatchLimitOne`
: Results are limited to the first item found; used as a value for the `kSecMatchLimit` attribute key.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

`kSecMatchLimitAll`
: An unlimited number of results may be returned; used as a value for the `kSecMatchLimit` attribute key.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

Item List Key

Key used to specify a list of items to search or add.

```
CFTypeRef kSecUseItemList;
```

Constants

`kSecUseItemList`
: Item list key.

  The corresponding value is of type `CFArrayRef`, where the array contains either `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, `SecIdentityRef`, or (for persistent item references) `CFDataRef` items. If provided, this array is treated as the set of all possible items to search (or to add if the function being called is `SecItemAdd` (page 42)). The items in the array must all be of the same type.

  Available in iOS 2.0 and later.

  Declared in `SecItem.h`.

Discussion
When this attribute is provided, no keychains are searched.

Search Results Constants

Return Type Keys

Keys used to specify the type of results that should be returned by the `SecItemCopyMatching` (page 43) or `SecItemAdd` (page 42) function.

```
CFTypeRef kSecReturnData;
CFTypeRef kSecReturnAttributes;
CFTypeRef kSecReturnRef;
CFTypeRef kSecReturnPersistentRef;
```

Constants

`kSecReturnData`

Return data attribute key.

The corresponding value is of type `CFBooleanRef`. A value of `kCFBooleanTrue` indicates that the data of an item should be returned in the form of a `CFDataRef`. For keys and password items, data is secret (encrypted) and may require the user to enter a password for access.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecReturnAttributes`

Return attributes attribute key.

The corresponding value is of type `CFBooleanRef`. A value of `kCFBooleanTrue` indicates that a dictionary of the (nonencrypted) attributes of an item should be returned in the form of a `CFDictionaryRef`.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecReturnRef`

Return reference attribute key.

The corresponding value is of type `CFBooleanRef`. A value of `kCFBooleanTrue` indicates that a reference should be returned. Depending on the item class requested, the returned references may be of type `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, `SecIdentityRef`, or `CFDataRef`.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecReturnPersistentRef`

Return persistent reference attribute key.

The corresponding value is of type `CFBooleanRef`. A value of `kCFBooleanTrue` indicates that a persistent reference to an item (`CFDataRef`) should be returned.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

Discussion

You can specify zero or more of these return types. If you specify more than one of these return types, Keychain Services returns the result as a `CFDictionaryRef` reference to a dictionary whose keys are the return types and whose alues are the requested data.

Value Type Keys

Keys used in the results dictionary for `SecItemCopyMatching` (page 43) or `SecItemAdd` (page 42), indicating the type of values returned. You can specify zero or more of these types depending on the function you are calling.

```
CFTypeRef kSecValueData;
CFTypeRef kSecValueRef;
CFTypeRef kSecValuePersistentRef;
```

Constants

`kSecValueData`

Data attribute key.

The corresponding value is of type `CFDataRef`. For keys and password items, the data is secret (encrypted) and may require the user to enter a password for access.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecValueRef`

Reference attribute key.

The corresponding value, depending on the item class requested, is of type `SecKeychainItemRef`, `SecKeyRef`, `SecCertificateRef`, or `SecIdentityRef`.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

`kSecValuePersistentRef`

Persistent reference attribute key.

The corresponding value is of type `CFDataRef`. The bytes in this `CFDataRef` can be stored by the caller and used on a subsequent invocation of the application (or even a different application) to retrieve the item referenced by it.

Available in iOS 2.0 and later.

Declared in `SecItem.h`.

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
`errSecSuccess`	0	No error. Available in iOS 2.0 and later.
`errSecUnimplemented`	-4	Function or operation not implemented. Available in iOS 2.0 and later.
`errSecParam`	-50	One or more parameters passed to the function were not valid. Available in iOS 2.0 and later.

Result Code	Value	Description
`errSecAllocate`	-108	Failed to allocate memory. Available in iOS 2.0 and later.
`errSecNotAvailable`	–25291	No trust results are available. Available in iOS 2.0 and later.
`errSecAuthFailed`	–25293	Authorization/Authentication failed. Available in iOS 4.2 and later.
`errSecDuplicateItem`	–25299	The item already exists. Available in iOS 2.0 and later.
`errSecItemNotFound`	–25300	The item cannot be found. Available in iOS 2.0 and later.
`errSecInteractionNotAllowed`	–25308	Interaction with the Security Server is not allowed. Available in iOS 2.0 and later.
`errSecDecode`	-26275	Unable to decode the provided data. Available in iOS 2.0 and later.

Randomization Services Reference

Framework: Security/Security.h

Declared in SecRandom.h

Overview

Randomization Services is an API that generates cryptographically secure random numbers.

Functions

SecRandomCopyBytes

Generates an array of cryptographically secure random bytes.

```
int SecRandomCopyBytes (
   SecRandomRef rnd,
   size_t count,
   uint8_t *bytes
);
```

Parameters

rnd

The random number generator object to use. Specify `kSecRandomDefault` to use the default random number generator.

count

The number of random bytes to return in the array pointed to by the `bytes` parameter.

bytes

The random bytes generated by the function.

Return Value

Returns `0` if the function completed successfully and `-1` if there was an error. Check the `errno` system variable for the error.

Discussion

This function reads from `/dev/random` to obtain an array of cryptographically-secure random bytes. For more information on the `/dev/random` random-number generator, see the manual page for random(4).

Availability

Available in iOS 2.0 and later.

Related Sample Code
CryptoExercise

Declared In
`SecRandom.h`

Data Types

SecRandomRef

Abstract Core Foundation-type object containing information about a random number generator.

```
typedef const struct __SecRandom * SecRandomRef;
```

Availability
Available in iOS 2.0 and later.

Declared In
`SecRandom.h`

Constants

Number Generator Default

Indicates the default random number generator.

```
const SecRandomRef kSecRandomDefault;
```

Constants
`kSecRandomDefault`
When passed to the `SecRandomCopyBytes` (page 71) function as the random number generator reference, this constant indicates that the default number generator should be used.

This constant is a synonym for `NULL`.

Available in iOS 2.0 and later.

Declared in `SecRandom.h`.

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.

Document Revision History