Certificate, Key, and Trust Services Reference

Security

2010-09-01

Contents

Tables

Certificate, Key, and Trust Services Reference

Framework:	Security/Security.h
Declared in	SecBase.h SecCertificate.h SecIdentity.h SecImportExport.h SecKey.h SecPolicy.h SecTrust.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 12)
: Returns the unique identifier of the opaque type to which a `SecCertificate` object belongs.

`SecIdentityGetTypeID` (page 13)
: Returns the unique identifier of the opaque type to which a `SecIdentity` object belongs.

`SecKeyGetTypeID` (page 17)
: Returns the unique identifier of the opaque type to which a `SecKey` object belongs.

`SecPolicyGetTypeID` (page 21)
: Returns the unique identifier of the opaque type to which a `SecPolicy` object belongs.

`SecTrustGetTypeID` (page 26)
: Returns the unique identifier of the opaque type to which a `SecTrust` object belongs.

Managing Certificates

`SecCertificateCreateWithData` (page 11)
: Creates a certificate object from a DER representation of a certificate.

`SecCertificateCopyData` (page 10)
: Returns a DER representation of a certificate given a certificate object.

`SecCertificateCopySubjectSummary` (page 10)
: Returns a human-readable summary of a certificate.

Managing Identities

`SecPKCS12Import` (page 19)
: Returns the identities and certificates in a PKCS #12-formatted blob.

`SecIdentityCopyCertificate` (page 12)
: Retrieves a certificate associated with an identity.

`SecIdentityCopyPrivateKey` (page 13)
: Retrieves the private key associated with an identity.

Cryptography and Digital Signatures

`SecKeyGeneratePair` (page 15)
: Creates an asymmetric key pair.

`SecKeyRawSign` (page 18)
: Generates a digital signature for a block of data.

`SecKeyRawVerify` (page 19)
: Verifies a digital signature.

`SecKeyEncrypt` (page 14)
: Encrypts a block of plaintext.

`SecKeyDecrypt` (page 14)
: Decrypts a block of ciphertext.

`SecKeyGetBlockSize` (page 17)
: Gets the block length associated with a cryptographic key.

Managing Policies

`SecPolicyCreateBasicX509` (page 20)
: Returns a policy object for the default X.509 policy.

`SecPolicyCreateSSL` (page 21)
: Returns a policy object for evaluating SSL certificate chains.

Managing Trust

`SecTrustCopyExceptions` (page 21)
: Returns an opaque cookie containing exceptions to trust policies that will allow future evaluations of the current certificate to succeed.

`SecTrustSetExceptions` (page 29)
: Sets a list of exceptions that should be ignored when evaluating the certificate.

`SecTrustCreateWithCertificates` (page 23)
: Creates a trust management object based on certificates and policies.

`SecTrustEvaluate` (page 24)
: Evaluates trust for the specified certificate and policies.

`SecTrustSetAnchorCertificates` (page 27)
: Sets the anchor certificates used when evaluating a trust management object.

`SecTrustSetAnchorCertificatesOnly` (page 29)
: Reenables trusting built-in anchor certificates.

`SecTrustSetVerifyDate` (page 30)
: Sets the date and time against which the certificates in a trust management object are verified.

`SecTrustGetVerifyTime` (page 27)
: Gets the absolute time against which the certificates in a trust management object are verified.

`SecTrustCopyPublicKey` (page 22)
: Returns the public key for a leaf certificate after it has been evaluated.

`SecTrustGetCertificateCount` (page 26)
: Returns the number of certificates in an evaluated certificate chain.

`SecTrustGetCertificateAtIndex` (page 25)
: 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 11)

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 11)

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 10)

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 31) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecCertificateRef` (page 31) 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 37).

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 37).

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 31) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecIdentityRef` (page 31) 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 32). 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 37).

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 32). 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 37).

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 37).

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 32) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecKeyRef` (page 32) 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 32). 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 37).

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 32). 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 37).

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 36).

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 37).

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 32) object.

Discussion

This function returns a value that uniquely identifies the opaque type of a `SecPolicyRef` (page 32) 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 29), 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 24) for that certificate will return `kSecTrustResultProceed` (page 34).

Note: If a new error occurs that did not occur when this function was called originally, the subsequent call to `SecTrustEvaluate` (page 24) can still fail. For example, if the certificate expires between calling `SecTrustCopyExceptions` and `SecTrustEvaluate` (page 24), 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 23) 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 24) 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 37).

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 24) 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 27)). 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 23) 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 34) for descriptions of possible values.

Return Value

A result code. See "Certificate, Key, and Trust Services Result Codes" (page 37).

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 30)) 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 27)).

By default, `SecTrustEvaluate` uses the current date and time when verifying a certificate. However, you can call the `SecTrustSetVerifyDate` (page 30) 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 23) function to create a trust management object and the `SecTrustEvaluate` (page 24) 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 26) 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 24) 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 23) function to create a trust management object and the `SecTrustEvaluate` (page 24) function to evaluate the certificate chain.

Return Value
The number of certificates in the certificate chain.

Discussion
You must call the `SecTrustEvaluate` (page 24) 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 32) object.

Discussion
This function returns a value that uniquely identifies the opaque type of a `SecTrustRef` (page 32) 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 23) 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 30) 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 30) function for the same trust management object on another thread.

Availability

Available in iOS 2.0 and later.

See Also

`SecTrustSetVerifyDate` (page 30)

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 23) 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 24) 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 37).

Discussion

The `SecTrustEvaluate` (page 24) 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 24) 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 24) 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 29) 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 29)

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 23) function to create a trust management object.

`anchorCertificatesOnly`
: If `true`, disables trusting any anchors other than the ones passed in with the `SecTrustSetAnchorCertificates` (page 27) 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 37).

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 24) 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 27)

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 21).

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 24) 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 23) 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 37).

Discussion

By default, the `SecTrustEvaluate` (page 24) 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 24) 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 27)

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 18) and `SecKeyRawVerify` (page 19) 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 18) and `SecKeyRawVerify` (page 19) 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 18) and `SecKeyRawVerify` (page 19) 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 15) 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 24) 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 24) 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 24) 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 24) 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 24) 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 24) again. This value may be returned by the `SecTrustEvaluate` (page 24) 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 24) 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 24) function. This value may be returned by the `SecTrustEvaluate` (page 24) 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 24) 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 19) 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 19) 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 24) 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.

AppleX509TP Trust Policies

The trust policies implemented by the AppleX509TP CDSA module are shown in Table A-1. A pointer to a policy-specific option structure is placed in `CSSM_FIELD.FieldValue.Data`; this field is one of the fields in `CSSM_TP_CALLERAUTH_CONTEXT.Policy.PolicyIds` array.

Table A-1 AppleX509TP trust policies

Policy	OID	Options	Description
Apple Basic	`CSSMOID_APPLE_-X509_BASIC`	None	Basic X509-style certificate evaluation
SSL	`CSSMOID_APPLE_TP_SSL`	`CSSM_APPLE_TP_-SSL_OPTIONS`	Basic X509 plus host name verification per RFC 2818
SMIME	`CSSMOID_APPLE_-TP_SMIME`	`CSSM_APPLE_TP_-SMIME_OPTIONS`	Basic X509 plus email address verification and KeyUsage enforcement per RFC 2632
Extensible Authentication Protocol (EAP)	`CSSMOID_APPLE_TP_EAP`	`CSSM_APPLE_TP_-SSL_OPTIONS`	Functionally identical to SSL policy. A separate OID is provided to facilitate per-policy, per-certificate trust settings using the SecTrust mechanism
CRL Revocation	`CSSMOID_APPLE_TP_-REVOCATION_CRL`	`CSSM_APPLE_TP_-CRL_OPTIONS` (see option flags below)	Revocation using certificate revocation lists

CRL Policy Options

Certificate revocation list policy options are defined in the following structure:

```
typedef uint32 CSSM_APPLE_TP_CRL_OPT_FLAGS;
enum {
    // require CRL verification for each cert; default is "try"
    CSSM_TP_ACTION_REQUIRE_CRL_PER_CERT = 0x00000001,
    // enable fetch from network
    CSSM_TP_ACTION_FETCH_CRL_FROM_NET  = 0x00000002
};
typedef struct {
    // CSSM_APPLE_TP_CRL_OPTS_VERSION
    uint32                           Version
    CSSM_APPLE_TP_CRL_OPT_FLAGS      CrlFlags;
    /*
    * When non-NULL, store CRLs fetched from net here.
    * This is most likely a pointer to one of the
    * CSSM_TP_CALLERAUTH_CONTEXT.DBList entries but that
    * is not a strict requirement.
    */
    CSSM_DL_DB_HANDLE_PTR              crlStore;
} CSSM_APPLE_TP_CRL_OPTIONS;
```

When the `CSSM_TP_ACTION_REQUIRE_CRL_PER_CERT` flag is set, a certificate is not valid unless every certificate in the certificate chain has been successfully verified using a certificate revocation list. This check is in addition to any other certificate-specific or policy-specific checks required for validation. When this flag is not set, CRLs are evaluated if they are available, but it is not an error if the trust policy module cannot find a CRL. In either case, the certificate is not considered valid if a CRL is found that indicates that any certificate in the certificate chain has been revoked.

Evaluating Multiple Policies

If multiple policies are specified, they are evaluated sequentially. In this case, the `VerificationAbortOn` field of the `CSSM_TP_CALLERAUTH_CONTEXT` structure specifies when to abort the verification process (see the "Trust Policy Services API" chapter in *Common Security: CDSA and CSSM, version 2 (with corrigenda)* from The Open Group (http://www.opengroup.org/security/cdsa.htm)). The AppleX509TP CDSA module supports the following values for the `VerificationAbortOn` field:

- `CSSM_TP_STOP_ON_NONE`; continue to subsequent policies if one policy evaluation fails
- `CSSM_TP_STOP_ON_FIRST_FAIL`; stop immediately if a policy evaluation fails
- `CSSM_TP_STOP_ON_POLICY`; treated as the same as `CSSM_TP_STOP_ON_FIRST_FAIL`

Document Revision History

This table describes the changes to *Certificate, Key, and Trust Services Reference*.

Date	Notes
2010-09-01	Added iOS 4 trust exception functions (SecTrustCopyExceptions and SecTrustSetExceptions).
2010-07-09	Made minor typographical fixes.
2010-04-06	Corrected typos.
2010-01-20	Corrected typos.
2009-10-12	Fixed several minor bugs.
2009-04-30	Updated for Mac OS X v10.6.
	This revision synchronizes the programming interface for Mac OS X with iOS v2.0 and adds information about concurrency.
2008-11-19	Updated to Mac OS X v 10.5.
	Trust settings functions and constants are new to this revision.
2008-06-25	Added constants and functions to sign documents, evaluate signatures, encrypt and decrypt data, return information about certificates, and import identities.
2005-03-03	Corrected the parameter list and description of SecKeyCreatePair.
	Corrected the parameter list and description of SecKeyCreatePair.
2004-05-27	Moved trusted applications functions to *Keychain Services Reference*.
2004-04-29	Corrected and expanded descriptions of functions.
	Added appendix listing AppleX509TP trust policies.
2003-05-15	Preliminary version of this document.