Macintosh File Signing

Apple Data Security Services

Application Interfaces (API)

Version 1.0b1
June 7, 1999

Macintosh File Signing overview

A **digitally signed file** is a file together with additional data that contains a secure hash of the file or portions of the file, and protected by a user's secret key. This data may be stored inside the file, or in a separate file.

Files signed using the Macintosh File Signing APIs will have additional resources added to the file containing the signature information. A signed Macintosh file will contain two new kinds of resources:

‘sig#’ - list of resources and data segments to include or exclude
‘sign’ - the signature data in PKCS#7 format

The ‘sig#’ resources specify what portions of the file should be included in the hash computed for the digital signature. The ‘sign’ resources are the signatures of a given ‘sig#’ resource.

Another type of resource, a signature mapping resource, of type 'sgm#' is used only by signing applications. This resource contains mappings between file types, creators, and default 'sig#' resources.

Using **Macintosh File Signing**

The following example demonstrates how a typical application might use the high-level Macintosh file signing interfaces to verify a digitally signed file.

```
OSStatus CheckAFile (FSSpec file)
{
      OSStatus status;
      SecOptions secOptions=kSecOptionProgress;
      SecSignerRef signer;
      Boolean alwaysShowUI=true;

      if (!KeychainManagerAvailable ())   // is it there?
         return (OSStatus)errKCNotAvailable;

      err=SecMacVerifyFileSimple(&file,secOptions,
         kSecTrustPolicyCodeSigning,&signer);
      SecMacDisplaySigner(signer,alwaysShowUI,&file);
      return (status);
}
```

Signing a Macintosh File

```
OSStatus SecMacSignFile (FSSpec *file, KCItemRef signingCertificate,
                         Handle sigH, SecOptions options,
                         SecProgressCallbackUPP progressProc,
                         void *userContext);
```

file	File to add signature resources to
signingCertificate	Certificate that has a matching signing key in one of the keychains specified.
sigH	Handle to 'sig#' resource. This should be obtained from the signing application's resources.
options	Show a display of progress
progressProc	A callback to a user-supplied progress routine. This may be set to NULL to use the default progress dialog.
userContext	If progressProc is not NULL, this is a user-supplied pointer that can retain state for the callback procedure

DESCRIPTION

Use this routine to add a digital signature to the specified file. The caller is responsible for releasing the signingCertificate using KCRelease.

RESULT CODE

noErr	0	No error.
paramErr	-50	One of the specified arguments is invalid
errKCNoDefaultKeychain	-25307	No default keychain could be found.
errKCItemNotFound	-25300	No matching certificate object was found.

```
OSStatus SecMacSignFileSimple (FSSpec *file, KCItemRef signingCertificate,
                         SecOptions options);
```

file	File to sign
signingCertificate	Certificate that has a matching signing key in one of the user's keychains.
options	Show a display of progress

DESCRIPTION

Use this routine to add a digital signature to the specified file. This routine is very similar to SecMacSignFile, except that default parameters are supplied internally.to add a digital signature to the specified file. An appropriate 'sig#' resource for the given file is found in the shared library. In addition, if kSecOptionProgress is set in *options*, the default progress dialog and callback will be used.

RESULT CODE

noErr	0	No error.
paramErr	-50	One of the specified arguments is invalid
errKCNoDefaultKeychain	-25307	No default keychain could be found.
errKCItemNotFound	-25300	No matching certificate object was found.

```
OSStatus SecMacLoadSigPound (FSSpec *file, Handle *resHandle);
```

`file`	`File that is being signed`
`resHandle`	`The 'sig#' resource appropriate for the file`

DESCRIPTION

Use this routine to determine load the proper 'sig#' resource for the specified file. The signature mapping resource ('sgm#') will be searched to get the correct resource ID. The handle returned in `resHandle` can be passed directly to `SecMacSignFile`. The caller is responsible for disposing of the memory using DisposeHandle. The calling application's resources will be searched for a valid 'sgm#' resource, which will be appended to the library's 'sgm#' resource before the search. This allows signing applications to override the built-in behavior.

RESULT CODE

`noErr`	0	No error.
`paramErr`	-50	One of the specified arguments is invalid
`resNotFound`	-192	No 'sig#' resource could be found.

```
OSStatus SecMacRemoveSignature (FSSpec *file, SecSignatureType
                    signatureType,SInt16 signatureToRemove);
```

file	File to remove signature resource from
signatureType	The type of signature to remove
signatureType	The resource ID of the signature to remove

DESCRIPTION

Use this routine to remove a 'sign' resource from the specified file. The signatureType is checked before removing. In release 1.0 of iSign, only one signature is validated, so this call will let signing applications remove an existing signature if the user wishes to re-sign the file.

RESULT CODE

noErr	0	No error.
paramErr	-50	One of the specified arguments is invalid
resNotFound	-192	No such signature resource could be found.

```
OSStatus SecMacRemoveAllSignatures (FSSpec *file, SecSignatureType
                    signatureType);
```

file	File to remove signature resource from
signatureType	The type of signature to remove
signatureType	The resource ID of the signature to remove

DESCRIPTION

Use this routine to remove all 'sign' resource from the specified file. The signatureType is checked before removing.

RESULT CODE

noErr	0	No error.
paramErr	-50	One of the specified arguments is invalid
resNotFound	-192	No such signature resource could be found.

```
OSStatus SecMacVerifyFile (FSSpec *file, SecOptions options,
                        SecProgressCallbackUPP progressProc, void
                        *userContext,SInt16 signatureToVerify, CFArrayRef
                        policyOIDs,KCVerifyStopOn stopOn, SecSignerRef *
                              signer);
```

file	File to verify signature of
options	Whether to add leaf certs, new anchors, or show a progress dialog
progressProc	A callback to a user-supplied progress routine. This may be set to NULL to use the default progress dialog.
userContext	If progressProc is not NULL, this is a user-supplied pointer that can retain state for the callback procedure
signatureToVerify	The resource ID of the signature to verify. The first signature in a file has a resource ID of 1.
policyOIDs	An array of policy OIDs that determine the trust policy. To get a pointer to an array of policy OIDs for Macintosh file signing, call SecMacGetDefaultPolicyOIDs.
stopOn	Use kPolicyKCStopOn for the default value. This will leave stop decisions up to the Trust Policy module.
signer	An opaque data structure containing information about the signer of the file, including the signer's certificate. This can be passed to other calls to display information about the signer. Pass in NULL if no information about the signer is needed.

DESCRIPTION

Use this routine to verify the digital signature to the specified file

RESULT CODE

noErr	0	No error.
paramErr	-50	One of the specified arguments is invalid
errKCNoDefaultKeychain	-25307	No default keychain could be found.

```
OSStatus SecMacVerifyFileSimple (FSSpec *file, SecOptions options,
                        SecTrustPolicy trustPolicy,SecSignerRef *signer);
```

file	File to verify signature of
options	Whether to bring display progress, add leaf certs and whether to add new anchors
trustPolicy	Specifies the trust level required for verification of a file. Only "kSecTrustPolicyCodeSigning" is supported in Release 1.0 of iSign
signer	An opaque data structure containing information about the signer of the file, including the signer's certificate. This can be passed to other calls to display information about the signer. Pass in NULL if no information about the signer is needed.

DESCRIPTION

Use this routine to verify a digital signature on the specified file. This routine is very similar to SecMacVerifyFile, except that default parameters are supplied internally.to find the first Apple-signed digital signature in the specified file. In addition, if kSecOptionProgress is set in *options*, the default progress dialog and callback will be used.

RESULT CODE

`noErr`	0	No error.
`paramErr`	-50	One of the specified arguments is invalid
`errKCNoDefaultKeychain`	-25307	No default keychain could be found.
`errSecVerifyFailed`	-24307	The file could not be validated.

`This is implemented as a call to SecMacVerifyFile with:`

`progressProc`	`NULL (so the default progress dialog is used)`
`userContext`	`NULL (the default progress dialog is used)`
`signatureToVerify`	`call SecMacFindSignatureToVerify with kSecSignatureTypeiSign`
`policyOIDs`	`SecMacGetDefaultPolicyOIDs(kSecTrustPolicyCodeSigning)`
`stopOn`	`kSecStopOnPolicy`

```
OSStatus SecMacHasSignature (FSSpec *file, SecSignatureType signatureType);
```

`file`	`File to test for signature`
`signatureType`	`The type of signature to check for`

DESCRIPTION

This routine provides a quick way to determine if a file contains a signature of a certain type. It does not verify the signature.

RESULT CODE

`false`	0	No signature of the specified type was found
`true`	1	A signature of the specified type was found

```
OSStatus SecMacFindSignatureToVerify (FSSpec *file,
      SecSignatureType signatureType, SInt16 *signatureToVerify);
```

`file`	`File to verify signature of`
`signatureType`	`The type of signature to check for`
`signatureToVerify`	`The resource ID of the signature is returned here`

DESCRIPTION

Use this routine to determine the resource ID for the first digital signature of the given type found in the specified file. The value returned in `signatureToVerify` can be passed directly to `SecMacVerifyFile`.

RESULT CODE

noErr	0	No error.
paramErr	-50	One of the specified arguments is invalid

```
CFArrayRef SecMacGetDefaultPolicyOIDs (SecTrustPolicy trustPolicy);
```

trustPolicy	One of kSecTrustPolicyCodeSigning or kSecTrustPolicyPersonalFileSigning

DESCRIPTION

Use this routine to get a pointer to an array of policy OIDs for Macintosh file signing. The output from this routine may be passed directly to `SecMacVerifyFile as the policyOIDs parameter`.

Constants

Values for the SecOptions flag

kSecOptionProgress	Show a progress indicator if an operation will take more than 2.5 seconds
kSecOptionShowVerifyUI	Show a dialog with the outcome of the verify operation
kSecOptionNeverShowUI	If set, don't show dialogs for adding certificates during a verify operation

Values for the SecSignatureType field

These constants are used internally, and are used in the fifth byte of the signature ('sign') resource to identify the type of the signature.

kSecSignatureTypeRawPKCS7	The signature data that follows is in raw PKCS7 format
kSecSignatureTypeiSign	The signature data that follows is in Macintosh file signing format
kSecSignatureTypeSMIME	The signature data that follows is in S/MIME format
kSecSignatureTypePGP	The signature data that follows is in PGP format (reserved, but not implemented)

Values for the SecTrustPolicy field

These constants are used internally by the Trust Policy module to define the conditions controlling termination of the verification process when a set of policies or conditions must be tested.

kSecTrustPolicyCodeSigning	Require a level 2 developer certificate signed by a root in the built-in root database
kSecTrustPolicyPersonalFileSigning	Any certificate signed by a valid root may be used

Progress Dialog callback functions

```
typedef CALLBACK_API(OSStatus,SecProgressCallbackProcPtr)
         (SecProgressCallbackInfo* info,void* userContext);

typedef STACK_UPP_TYPE(SecProgressCallbackProcPtr)SecProgressCallbackUPP;

pascal OSStatus DefaultSecProgressCallbackProc(SecProgressCallbackInfo* info, void*
         userContext);

struct SecProgressCallbackInfo
{
         UInt32   version;
         UInt32   bytesProcessed;
         UInt32   totalBytes;
         UInt32   itemsRemainingToSign;
         UInt32   totalItemsToSign;
         UInt32   secondsRemaining;
         UInt32   secondsElapsed;
         UInt32   microSecondsPerByte;
         Str255   fileName;
};
typedef struct SecProgressCallbackInfo SecProgressCallbackInfo;
```

Format of the ‘sig#’ resource

The 'sig#' resource may be found in any Macintosh file. It represents information regarding how a particular file is signed. It does not contain information regarding actual signatures. It has the following format:

```
Int16 majorVersion
Int16 minorVersion
String            componentName          // reserved for expansion
OSType            fileCreator
OSType            fileType
Int16 finderFlagMask
UInt8 cfrgProcessing                     // reserved for expansion
Bit               excludeCreator
Bit               excludeType
Bit               isResourceExcludeList
Bit               fill
Bit               fill
bit               fill
Bit               fill
Bit               fill
align long
{
      Int32       offset
      Int32       length
} List            dataForkBlocks
{
      OSType      resourceType
      Int16       baseResourceID
      Int16       endResourceID
      Int16       ResourceAttrMask;
      Int32       offset
      Int32       length
} List            resourceBlocks
```

Descriptions

```
fileCreator       The creator of the file
fileType          The type of the file
finderFlagMask    This is ANDed with the file's Finder flags before adding to
                  the digest
cfrgProcessing    This field is reserved for expansion
excludeCreator    Do not include the file's creator in the digest
excludeType       Do not include the file's type in the digest
isResourceExcludeListThe list of resource blocks is an exclusion list, i.e.
                  these resources will be excluded from the digest

baseResourceID    The lower bound for a range of resource IDs
endResourceID     The upper bound for a range of resource IDs
ResourceAttrMask  This is ANDed with each resource's attributes before adding
                  to the digest
```

Format of the ‘sign’ resource

The ‘sign’ resource contains the data repesenting the signer of the file. The `signatureType` field of the resource identifies the type of the remainder of the resource. The signatureType used by Macintosh file signing is `kSecMsgDomainiSign`.

```
Int16           majorVersion
Int16           minorVersion
UInt8           signatureType
UInt8           fill
Int16           sigListID   // res ID of corresponding ‘sig#’
void *          remainderOfData
```

Format of the ‘sgm#’ resource

The 'sgm#' resource is used only by signing applications to map file type and creator pairs to a default ‘sig#’ resource. It has the following format:

```
Int16           majorVersion
Int16           minorVersion
{
     OSType     fileType
     OSType     fileCreator
     Int16      sigPoundID
} list
```

When a file is presented for signing, the signing application should look for the file’s type and creator in the list. If a match is found, the corresponding sigPoundID is used to find a ‘sig#’ resource in the signing application that will be used to sign the given file. The code ‘****” may be used as a wild card to match any type or creator.

These resources are used by the SecMacSignFileSimple call to find the appropriate 'sig#' resource.

Signing a Macintosh File

```
OSStatus SecMacSignFile (FSSpec *file, KCItemRef signingCertificate,
                         Handle sigH, SecOptions options,
                         SecProgressCallbackUPP progressProc,
                         void *userContext);

OSStatus SecMacSignFileSimple (FSSpec *file, KCItemRef signingCertificate,
                         SecOptions options);
```

Signing support routines

```
OSStatus SecMacLoadSigPound (FSSpec *file, Handle *resHandle);

OSStatus SecMacRemoveSignature (FSSpec *file, SecSignatureType
                         signatureType,SInt16 signatureToRemove);

OSStatus SecMacRemoveAllSignatures (FSSpec *file, SecSignatureType
                         signatureType);
```

Verifying the signature on a Macintosh File

```
OSStatus SecMacVerifyFile (FSSpec *file, SecOptions options,
                         SecProgressCallbackUPP progressProc, void
                         *userContext,UInt16 signatureToVerify, CFArrayRef
                         policyOIDs,KCVerifyStopOn stopOn,SecSignerRef
                         *signer);

OSStatus SecMacVerifyFileSimple (FSSpec *file, SecOptions
                         options,SecTrustPolicy,SecSignerRef *signer);
```

Verification support routines

```
OSStatus SecMacHasSignature (FSSpec *file, SecSignatureType signatureType);

OSStatus SecMacFindSignatureToVerify (FSSpec *file, SecSignatureType
                         signatureType, SInt16 *signatureToVerify);

CFArrayRef SecMacGetDefaultPolicyOIDs (SecTrustPolicy trustPolicy);

OSStatus SecMacDisplaySigner (SecSignerRef *signer, Boolean
                         alwaysShowUI, FSSpec *file);
```

Data Types and constants

```
enum
{
   kSecOptionProgress         = 1 << 0,
   kSecOptionShowVerifyUI     = 1 << 1
} ;
typedef UInt32               SecOptions;

enum
{
   kSecSignatureTypeRawPKCS7  = 0,
   kSecSignatureTypeiSign     = 1,
   kSecSignatureTypeSMIME     = 2,
   kSecSignatureTypePGP       = 3         /* reserved but not implemented */
};
typedef UInt32               SecSignatureType;

enum
{
   kSecTrustPolicyCodeSigning                   = 0,
   kSecTrustPolicyPersonalFileSigning           = 1
};
typedef UInt32               SecTrustPolicy;
```