Framework

Security

Secure the data your app manages, and control access to your app.

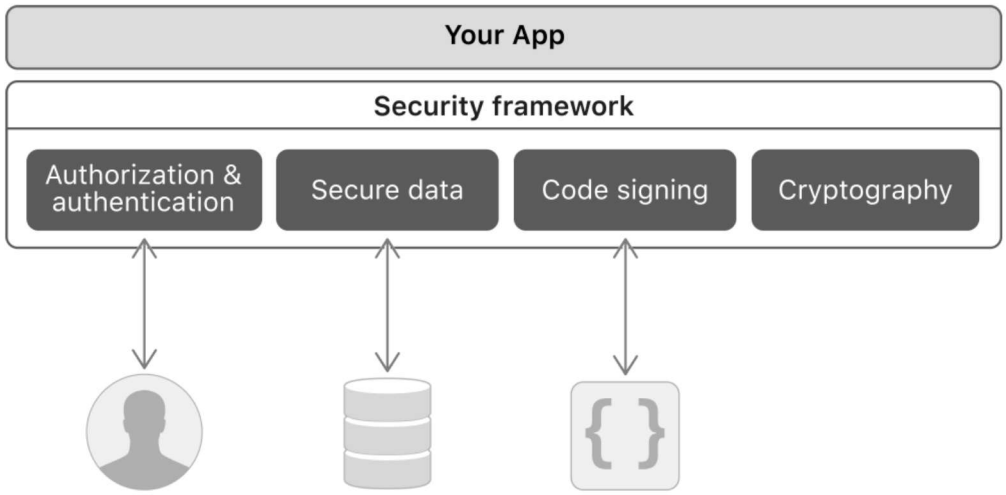
iOS 2.0+ iPadOS 2.0+ Mac Catalyst 13.0+ macOS 10.0+ tvOS 9.0+ visionOS 1.0+ watchOS 2.0+

Overview

Use the Security framework to protect information, establish trust, and control access to software. Broadly, security services support these goals:

- Establish a user's identity (authentication) and then selectively grant access to resources (authorization).
- Secure data, both on disk and in motion across a network connection.
- Ensure the validity of code to be executed for a particular purpose.

As shown in the image below, you can also use lower level cryptographic resources to create new secure services. Cryptography is difficult and the cost of bugs typically so high that it's rarely a good idea to implement your own cryptography solution. Rely on the Security framework when you need cryptography in your app.



Note

Always use the highest level API that meets your needs. The Security framework is not always your best option. For example, to conduct secure network communications, start by considering the Foundation framework's URL Loading System, which builds on the Security framework. Only if your app requires lower level access to security protocol functions would you use the secure transport API directly.

Topics

Essentials

Security updates

Learn about important changes to Security.

Authorization and authentication

Password AutoFill

Streamline your app's login and onboarding procedures.

Shared Web Credentials

Share credentials between iOS apps and their website counterparts.

Authorization Services

Access restricted areas of the operating system, and control access to particular features of your macOS app.

- Authorization Plug-ins

  Extend the authorization services API by creating plug-ins that can participate in authorization decisions.

- Sessions

  Manage login, authorization, and security sessions in macOS.

- One-time codes

  Streamline entry of authentication and recovery codes.

Secure data

- Keychain services

  Securely store small chunks of data on behalf of the user.

- Preventing Insecure Network Connections

  Enforce secure network links in your app by relying on App Transport Security.

Secure code

- Code Signing Services

  Examine and validate signed code running on the system.

- Notarizing macOS software before distribution

  Give users even more confidence in your macOS software by submitting it to Apple for notarization.

- Preparing your app to work with pointer authentication

  Test your app against the arm64e architecture to ensure that it works seamlessly with enhanced security features.

- App Sandbox

  Restrict access to system resources and user data in macOS apps to contain damage if an app becomes compromised.

- Hardened Runtime
  Manage security protections and resource access for your macOS apps.

- Disabling and Enabling System Integrity Protection
  Disable system protections only temporarily during development to test drivers, kernel extensions, and other low-level code.

- Using the latest code signature format
  Update legacy app code signatures so your app runs on current OS releases.

- Updating Mac Software
  Implement Mac software updates without causing code-signing crashes.

- TN3125: Inside Code Signing: Provisioning Profiles
  Learn how provisioning profiles enable third-party code to run on Apple platforms.

Launch environment constraints

- Applying launch environment and library constraints
  Limit the libraries your process loads, and the situations where it runs.

- Defining launch environment and library constraints
  Restrict your app's components to their expected contexts.

- Constraining a tool's launch environment
  Improve the security of your macOS app by limiting the ways its components can run.

Cryptography

- Complying with Encryption Export Regulations
  Declare the use of encryption in your app to streamline the app submission process.

- Certificate, Key, and Trust Services
  Establish trust using certificates and cryptographic keys.

- Cryptographic Message Syntax Services
  Cryptographically sign and encrypt S/MIME messages.

- Randomization Services

  Generate cryptographically secure random numbers.

- Security Transforms

  Perform cryptographic functions like encoding, encryption, signing, and signature verification.

- ASN.1

  Encode and decode Distinguished Encoding Rules (DER) and Basic Encoding Rules (BER) data streams.

Result codes

- Security Framework Result Codes

  Evaluate result codes common to many Security framework functions.

Legacy interfaces

- Common Security Services Manager

  A set of open source modules underpinning the legacy implementation of the Security framework.

- Secure Transport

  Secure network communication using standardized transport layer security mechanisms.

- Secure Download

  Implement Apple's Secure Download System in macOS.

- Security legacy reference

  Learn about legacy APIs.

Reference

- Security Structures

- Security Constants

Security Functions

Security Data Types

Variables

```
var CSSM_APPLE_PRIVATE_CSPDL_CODE_28: Int
```

```
var TLS_ECDHE_PSK_WITH_CHACHA20_POLY1305_SHA256: SSLCipherSuite
```

```
var errSecMissingQualifiedCertStatement: OSStatus
```

```
let kSecPolicyAppleEAPClient: CFString
```

```
let kSecPolicyAppleEAPServer: CFString
```

```
let kSecPolicyAppleIPSecClient: CFString
```

```
let kSecPolicyAppleIPSecServer: CFString
```

```
let kSecPolicyAppleSSLClient: CFString
```

```
let kSecPolicyAppleSSLServer: CFString
```

```
let kSecTrustQCStatements: CFString
```

```
let kSecTrustQWACValidation: CFString
```

Functions

```
func SecIdentityCreate(CFAllocator?, SecCertificate, SecKey) -> Sec
Identity?
```

```
func sec_protocol_metadata_copy_negotiated_protocol(sec_protocol
_metadata_t) -> UnsafePointer<CChar>?
```

```
func sec_protocol_metadata_copy_server_name(sec_protocol_metadata_t) -
> UnsafePointer<CChar>?
```

Type Aliases

```
typealias CE_DataType
```

```
typealias CE_ExtendedKeyUsage

typealias CE_GeneralNameType
```