Protecting Secrets with the Keychain

The easier way to keep secrets

Session 709

Perry The Cynic
Security Architect

What Keychains Are

How to Use Them

iOS Specifics

OS X Specifics

References

Mission

- Store small secrets (passwords, codes, etc.) securely
- Limit access to particular users and applications
- Protect against offline attack
- Store secrets under user control
- Standard high-level solution

Features

- A very specialized database
- Metadata (attributes)
- Protected data (payload)
- Efficient search by metadata
- Optimized for small payload and single access
  - Not a good choice for thousands or megabytes of items

Why Bother?

- Tie to user password and hardware
- Access control
- Controlled sharing between Apps
- Files are easily scanned, copied, analyzed
- Stuff leaks
- Security is hard

The Short Form

Just do this

The No-Brainer Calls

Item creation

```
NSData* secret = [@”top secret” dataWithEncoding:NSUTF8StringEncoding];
NSDictionary* query = @{
     (id)kSecClass: (id)kSecClassGenericPassword,
     (id)kSecAttrService: @”myservice”,
     (id)kSecAttrAccount: @”account name here”,
     (id)kSecValueData: secret,
};
      OSStatus status = SecItemAdd((CFDictionaryRef)query, NULL);
```

The No-Brainer Calls

Item creation

```
NSData* secret = [@”top secret” dataWithEncoding:NSUTF8StringEncoding];
NSDictionary* query = @{
     (id)kSecClass: (id)kSecClassGenericPassword,
     (id)kSecAttrService: @”myservice”,
     (id)kSecAttrAccount: @”account name here”,
     (id)kSecValueData: secret,
};
      OSStatus status = SecItemAdd((CFDictionaryRef)query, NULL);
```

- Cannot create duplicate item in keychain

The No-Brainer Calls

Item lookup

```
NSDictionary* query = @{
     (id)kSecClass: (id)kSecClassGenericPassword,
     (id)kSecAttrService: @”myservice”,
     (id)kSecAttrAccount: @”account name here”,
     (id)kSecReturnData: @YES,
};
```

```
NSData *data = NULL;
      OSStatus status = SecItemCopyMatching((CFDictionaryRef)query,
     (CFTypeRef*)&data);
```

The No-Brainer Calls

Other basics

```
NSDictionary* query = @{
     (id)kSecClass: (id)kSecClassGenericPassword,
     (id)kSecAttrService: @”myservice”,
     (id)kSecAttrAccount: @”account name here”,
};
NSDictionary* changes = @{ ... };
     OSStatus status = SecItemUpdate((CFDictionaryRef)query,
    (CFDictionaryRef)changes);
OSStatus status = SecItemDelete((CFDictionaryRef)query);
```

The No-Brainer Calls

Other basics

```
NSDictionary* query = @{
     (id)kSecClass: (id)kSecClassGenericPassword,
     (id)kSecAttrService: @”myservice”,
     (id)kSecAttrAccount: @”account name here”,
};
NSDictionary* changes = @{ ... };
      OSStatus status = SecItemUpdate((CFDictionaryRef)query,
     (CFDictionaryRef)changes);
OSStatus status = SecItemDelete((CFDictionaryRef)query);
```

- Use Update, not Delete and Add

The “Memory” Keychain Workflow

The “Memory” Keychain Workflow

(Pseudo code)

```
NSData* password = nil;
if (SecItemCopyMatching(..., &password) == noErr) {
    if (password works) {
        great!
```

The “Memory” Keychain Workflow

(Pseudo code)

```
NSData* password = nil;
if (SecItemCopyMatching(..., &password) == noErr) {
    if (password works) {
        great!
} else {
    password = get from user;
    if (password works) {
        SecItemAdd(...);   // save it for next time
    }
}
```

The “Memory” Keychain Workflow

(Pseudo code)

```
NSData* password = nil;
if (SecItemCopyMatching(..., &password) == noErr) {
    if (password works) {
        great!
     } else {
        password = get a better one;
        if (that password worked better) {
            SecItemUpdate(...);
        }
    }
} else {
    password = get from user;
    if (password works) {
        SecItemAdd(...);   // save it for next time
    }
}
```

Considerations

- Try use before storing
- Distinguish bad passwords from bad environment
- Deal with external changes
- How valuable/retrievable is that password?
- Always have a fallback procedure

Put Secrets Into the Item Value

- Interpret “secret” liberally
  - Passwords, PINs, codes
  - Account numbers
  - What does the user want to hide?
- Attributes
  - Public identifiers, names
  - Be careful
  - If needed, make something up

Handling Retrieved Secrets

- Use and purge
- Do not keep secrets in memory
- Do not save or send

Keychains in iOS

Basics

- SecItem API does it all
- Tied to passcode and hardware (data protection)

Sharing Items

- Controlled by keychain-access-groups entitlement

Sharing Items

- Controlled by keychain-access-groups entitlement
- Values restricted by profiles and store policy

Sharing Items

- Controlled by keychain-access-groups entitlement
- Values restricted by profiles and store policy
  - Use your TEAMID prefix

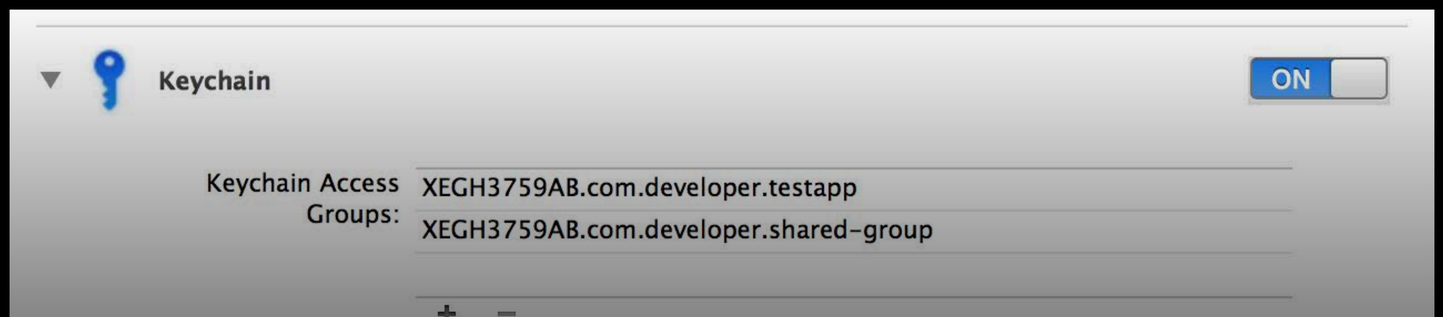
Sharing Items

iOS

- Controlled by keychain-access-groups entitlement
- Values restricted by profiles and store policy
  - Use your TEAMID prefix

Sharing Items

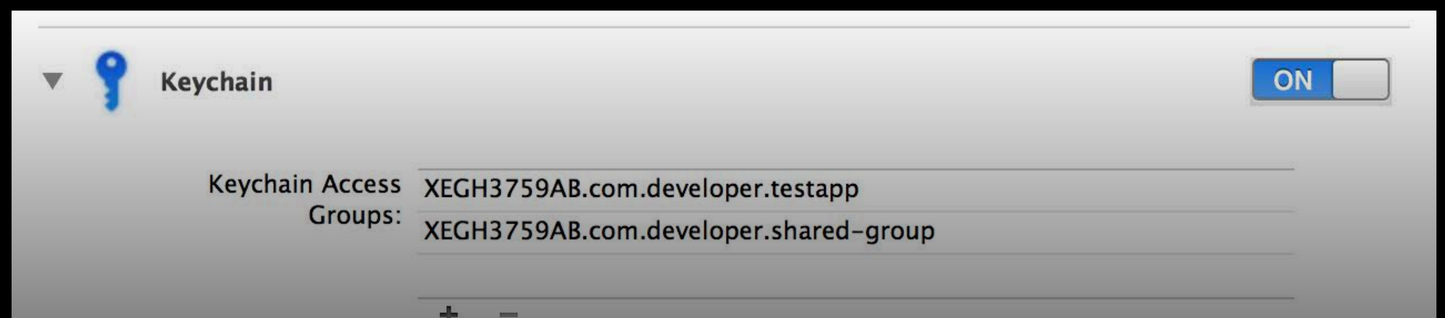
- Controlled by keychain-access-groups entitlement
- Values restricted by profiles and store policy
  - Use your TEAMID prefix



Sharing Items

- Controlled by keychain-access-groups entitlement
- Values restricted by profiles and store policy
  - Use your TEAMID prefix

```
$ codesign --display --entitlements=:- My.App
Executable=/Applications/Mail.app/Contents/MacOS/Mail
    ...
<plist version="1.0">
<dict>
    <key>keychain-access-groups</key>
    <array>
<string>XEGH3759AB.com.developer.testy</string>
    ...
```



Keychains and Data Protection

Keychains and Data Protection


```
(id)kSecAttrAccessible: kSecAttrAccessibleWhenUnlocked,
```

Keychains and Data Protection


```
(id)kSecAttrAccessible: kSecAttrAccessibleWhenUnlocked,
(id)kSecAttrAccessible: kSecAttrAccessibleAfterFirstUnlock,
```

Keychains and Data Protection


```
(id)kSecAttrAccessible: kSecAttrAccessibleWhenUnlocked,
(id)kSecAttrAccessible: kSecAttrAccessibleAfterFirstUnlock,
```

- Stick with `kSecAttrAccessibleWhenUnlocked` whenever possible
- Use `kSecAttrAccessibleAfterFirstUnlock` to use from a locked phone
- Consider storing (weaker) derived secrets for background use

Backup and Migration

- Keychain data is backed up with system
- Restore to same device is complete
- Migration requires encrypted backups
- Data protection rules apply
  - Per-device items are not migrated

Debugging

Debugging

- There is no tool for inspecting the iOS keychain

Debugging

- There is no tool for inspecting the iOS keychain
- Add in-app test/debug code

Debugging

- There is no tool for inspecting the iOS keychain
- Add in-app test/debug code
  - If you ship it, tell App Review

Debugging

- There is no tool for inspecting the iOS keychain
- Add in-app test/debug code
  - If you ship it, tell App Review
- Argument dictionaries do double duty

Debugging

- There is no tool for inspecting the iOS keychain
- Add in-app test/debug code
  - If you ship it, tell App Review
- Argument dictionaries do double duty
  - Attributes and selectors

Debugging

- There is no tool for inspecting the iOS keychain
- Add in-app test/debug code
  - If you ship it, tell App Review
- Argument dictionaries do double duty
  - Attributes and selectors
  - Operational instructions

Keychains in OS X

Basics

- SecItem API (preferred)
- SecKeychain legacy API (if needed)
- Keychains are files
- Tied to password and POSIX permissions
  - Unlocked with login password

Sharing Items

- Keychains are shared through the file system
  - login.keychain—per user
  - System.keychain—entire system
- Each item has an Access Control List (ACL)
- Set access on item creation
  - Change is possible, but discouraged

Sharing Items

Managing Keychain Item ACLs

- List of applications
  - …by code signing identity
  - SecAccess/SecACL/ SecTrustedApplication API
- Automatic prompt for any other caller
  - …with option to update the ACL
- System.keychain dialogs require admin access

Sharing Items

Managing Keychain Item ACLs

- List of applications
  - …by code signing identity
  - SecAccess/SecACL/ SecTrustedApplication API
- Automatic prompt for any other caller
  - …with option to update the ACL
- System.keychain dialogs require admin access

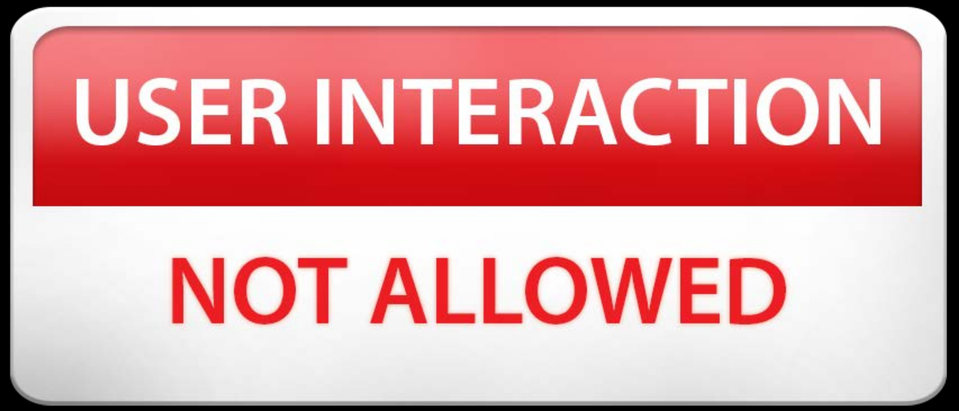
Sharing Items

Managing Keychain Item ACLs

- List of applications
  - …by code signing identity
  - SecAccess/SecACL/ SecTrustedApplication API
- Automatic prompt for any other caller
  - …with option to update the ACL
- System.keychain dialogs require admin access

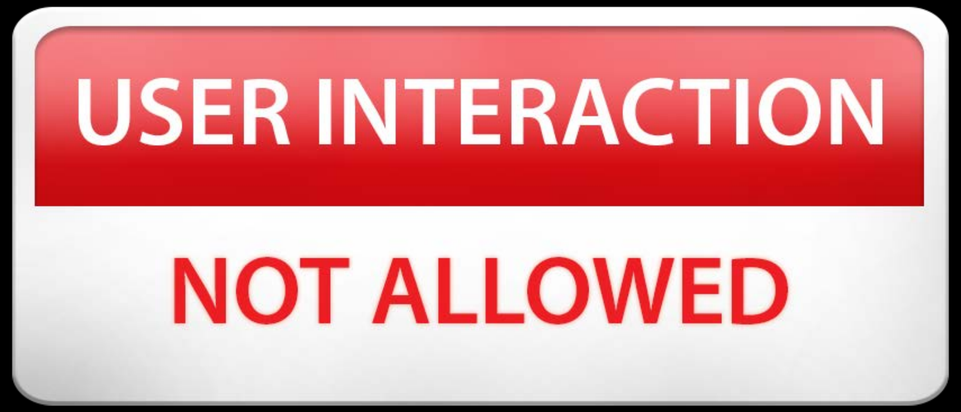
Security Prompts

Security Prompts



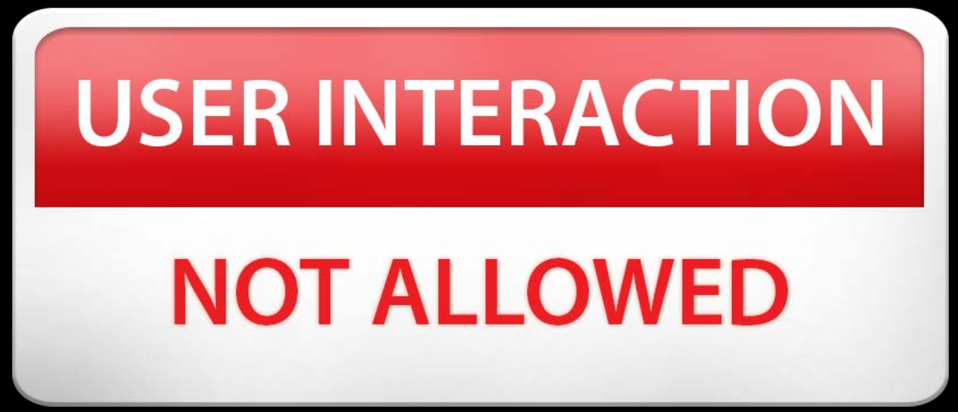
Security Prompts

- OS X will prompt the user to
  - Unlock a keychain
  - Confirm access for new applications



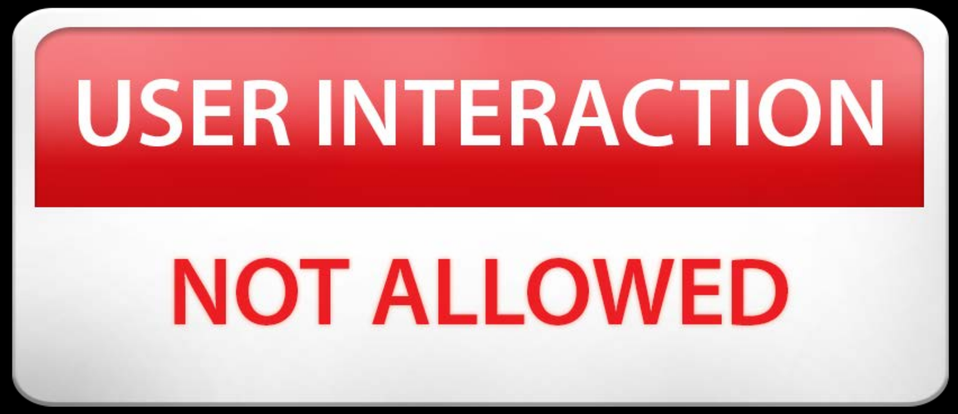
Security Prompts

- OS X will prompt the user to
  - Unlock a keychain
  - Confirm access for new applications
- This will fail if the login came from a nongraphic source



Security Prompts

- OS X will prompt the user to
  - Unlock a keychain
  - Confirm access for new applications
- This will fail if the login came from a nongraphic source
- See security(1)

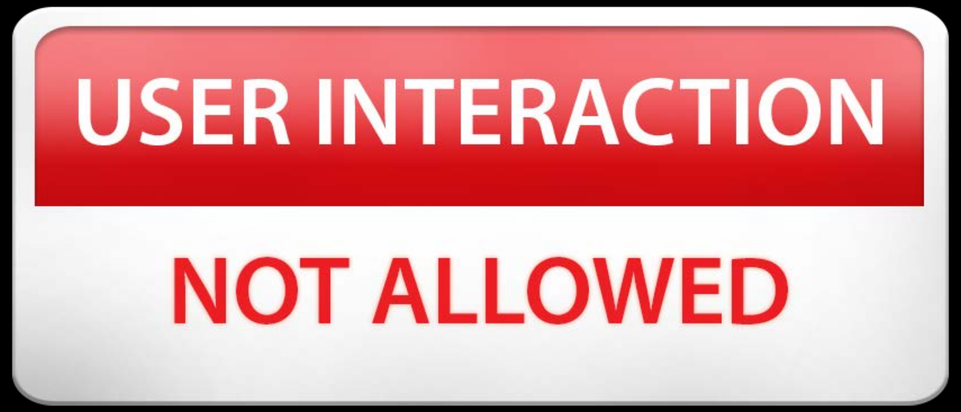


Backup and Migration

- Keychains are files
- Any backup strategy will work
- Migration Assistant preserves keychains

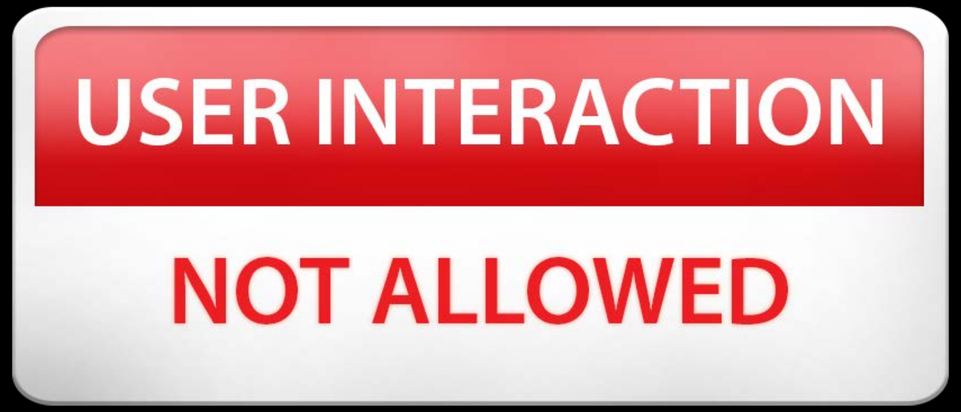
Keychain for Daemons

Keychain for Daemons



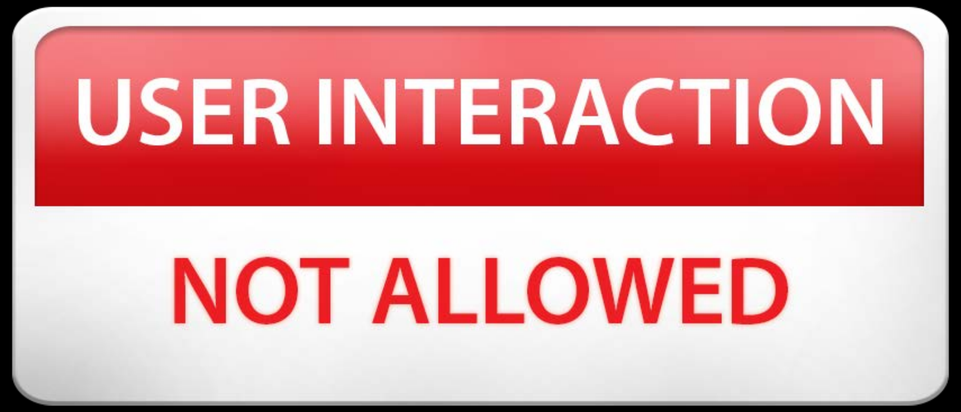
Keychain for Daemons

- System.keychain is the default keychain for daemons



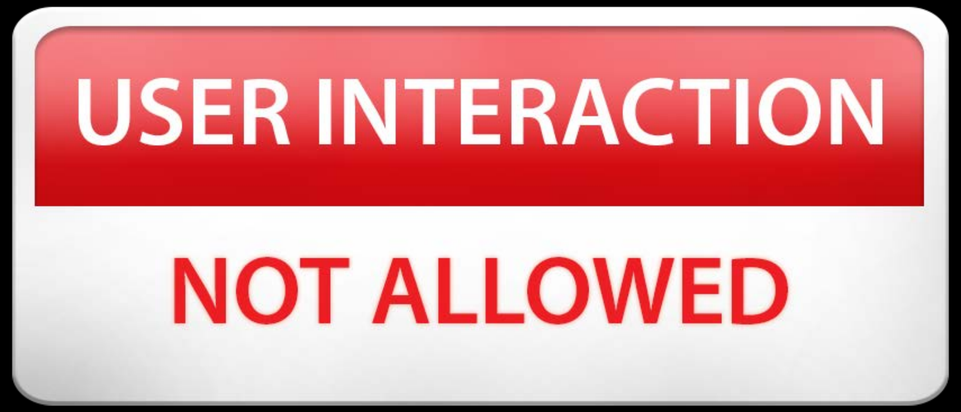
Keychain for Daemons

- System.keychain is the default keychain for daemons
- No access to “user” keychain (which?)



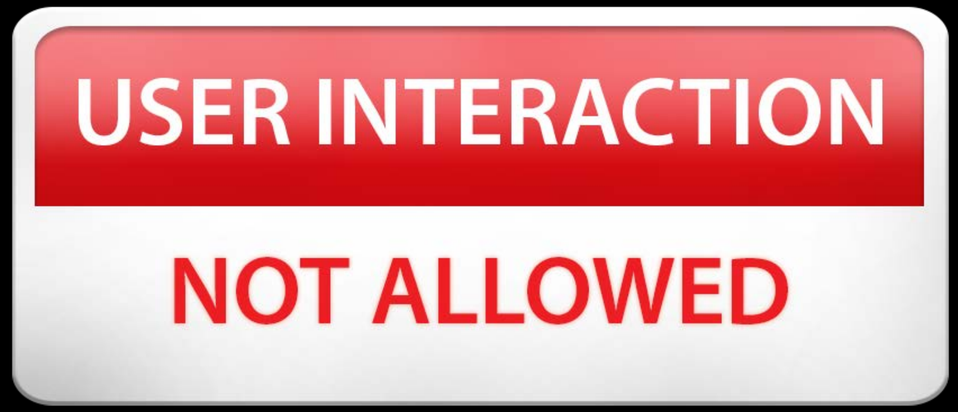
Keychain for Daemons

- System.keychain is the default keychain for daemons
- No access to “user” keychain (which?)
- Write requires root (POSIX)



Keychain for Daemons

- System.keychain is the default keychain for daemons
- No access to “user” keychain (which?)
- Write requires root (POSIX)
- Daemons cannot display keychain dialogs



Debugging

- security(1) is your friend

```
$ security show-keychain-info
$ security dump-keychain
$ security find-generic-password
$ security help
```

References

Keys and Certificates

- SecKey
- SecCertificate
- SecIdentity

```
(id)kSecAttrSynchronizable: @YES,
```

```
(id)kSecAttrSynchronizable: @YES,
```

iCloud Keychain

```
(id)kSecAttrSynchronizable: @YES,
```

iCloud Keychain

```
(id)kSecAttrSynchronizable: @YES,
```

- Sync items between iOS and OS X devices
- Add to all basic SecItem calls
- Some restrictions apply

Summary

- Use keychain APIs to easily store small secrets
- Don’t keep your secrets or store them elsewhere
- Keep it simple
  - Use SecItem APIs
  - Don’t get fancy—fancy is dangerous

More Information

Paul Danbold
CoreOS Technologies Evangelist
danbold@apple.com

Documentation

Keychain Services Documentation
http://developer.apple.com/library/mac/#documentation/Security/Conceptual/keychainServConcepts

Technical Q&A QA1745: Making Certificates and Keys Available To Your App
https://developer.apple.com/library/ios/#qa/qa1745/_index.html%23//apple_ref/doc/uid/DTS40011636

iOS Security White Paper
http://images.apple.com/iphone/business/docs/iOS_Security_Oct12.pdf

Apple Developer Forums

http://devforums.apple.com

Labs

Keychain and Data Protection Security Lab	Core OS Lab Wednesday 4:30PM
Security Lab	Core OS Lab Thursday 2:00PM
Privacy & Security Lab	Core OS Lab Friday 10:15AM

WWDC2013