Securing iOS Applications

Protecting users' data on the network and on the device

Session 208

Conrad Sauerwald
iOS Security Snarkitect

Andrew R. Whalley
iOS Security Manager

Michael Brouwer
iOS Security Architect

Introduction

- Embedded Data Security team
  - Any time a key is used to protect user data
    - Data Protection, Keychain, Secure Transport, CMS
  - Design and build solutions for internal clients
  - High-level API for 3rd-party applications

What We Will Demonstrate Today

- Test-drive an application through a bad neighborhood
  - What can happen
  - Why it matters
  - How you can avoid it
- Build, test, fix, repeat

Oversharing.app

Simple device-to-device photo-sharing app

- Discover other users on the local network with Bonjour
- Take pictures on one device, have them appear on another
- Source code available

Why Secure Oversharing.app?

- Users are entrusting their data to your app
  - You cannot foresee to what uses your app will be put
  - Assume that people care about their data. Protect it.
- Scary and abstract terms in the news
  - Brute force, man in the middle…
- Threats are very real
  - Losing a device at a conference
  - Connecting to a network

How iOS APIs Can Help

What we will be covering

- Securing network connections
- Protecting data
- Protecting secrets

Securing Network Connections

Securing Network Connections

- API
  - High-level API in Foundation: HTTP(S)…socket
  - Low-level API in SecureTransport: DTLS/TLS, data callback
- Application
  - HTTP GET to register
  - HTTP POST to publish pictures

Oversharing.app Networking

Application registers interest via GET

```
NSString *server_url = @"http://1.3.3.7:1337/register/name/";
NSURLRequest *request = [NSURLRequest requestWithURL:
                            [NSURL URLWithString: server_url ]];

[[NSURLConnection alloc] initWithRequest: request delegate: self];

// @interface MainViewController : UIViewController <NSURLConnectionDelegate>
- (void)connection:(NSURLConnection *)c didReceiveData:(NSData *)data
- (void)connection:(NSURLConnection *)c didFailWithError:(NSError *)error
```

Oversharing.app Networking

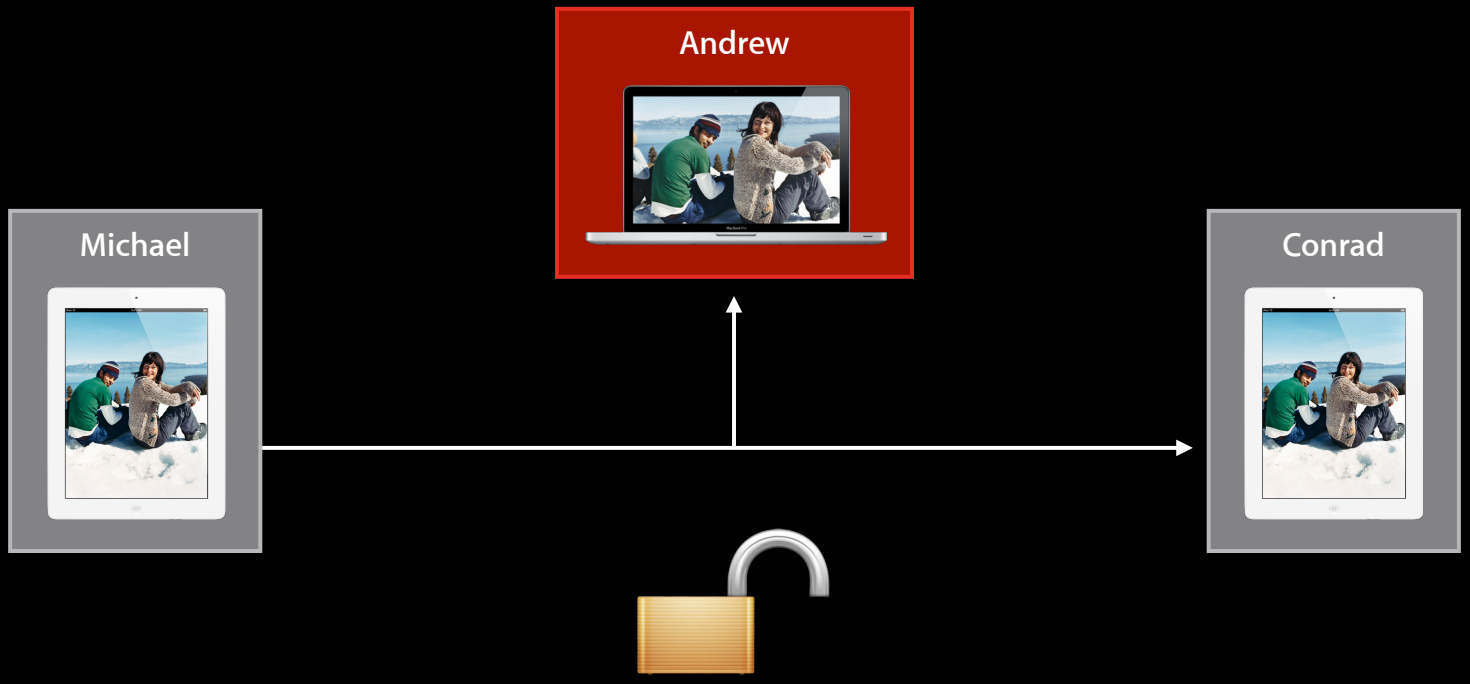
Publish picture via POST

```
NSString * publish_url = @”http://1.3.3.7:1334/publish/”;
NSMutableURLRequest *request = [NSMutableURLRequest requestWithURL:
                                    [NSURL URLWithString: publish_url]];
[request setHTTPBody:image_data];
[request setHTTPMethod:@"POST"];
[[NSURLConnection alloc] initWithRequest:request delegate:self];
```

Demo

Network Interception

What does the attacker see?



Using Transport Layer Security (TLS)

Foundation client

- One letter change, note the “https” in the URL

```
// “After” code
NSString *server_url = @”https://1.3.3.7:1337/{register|publish}/”;
```

Using Transport Layer Security (TLS)

SecureTransport server—setup

- Context, callbacks, and certificates

```
SSLContextRef ctx = NULL;
SSLNewContext(server, &ctx)
SSLSetIOFuncs(ctx,
    (SSLReadFunc)socket_read, (SSLWriteFunc)socket_write);
SSLSetConnection(ctx, (SSLConnectionRef)sock), out);

// TODO: deal with certificates later
// SSLSetCertificate(ctx, certs);
```

Using Transport Layer Security (TLS)

SecureTransport server—operation

- Open, read/write, close

```
// TLS handshake negotiates authentication and encryption
status = SSLHandshake(ctx);

// Read decrypted data from peer
status = SSLRead(ctx, buffer, sizeof(buffer), &bytes_read);

// Write encrypted data to peer
status = SSLWrite(ctx, buffer, sizeof(buffer), &bytes_written);

// Shutdown TLS
SSLClose(ctx);
```

TLS Without Certificates

Winning?

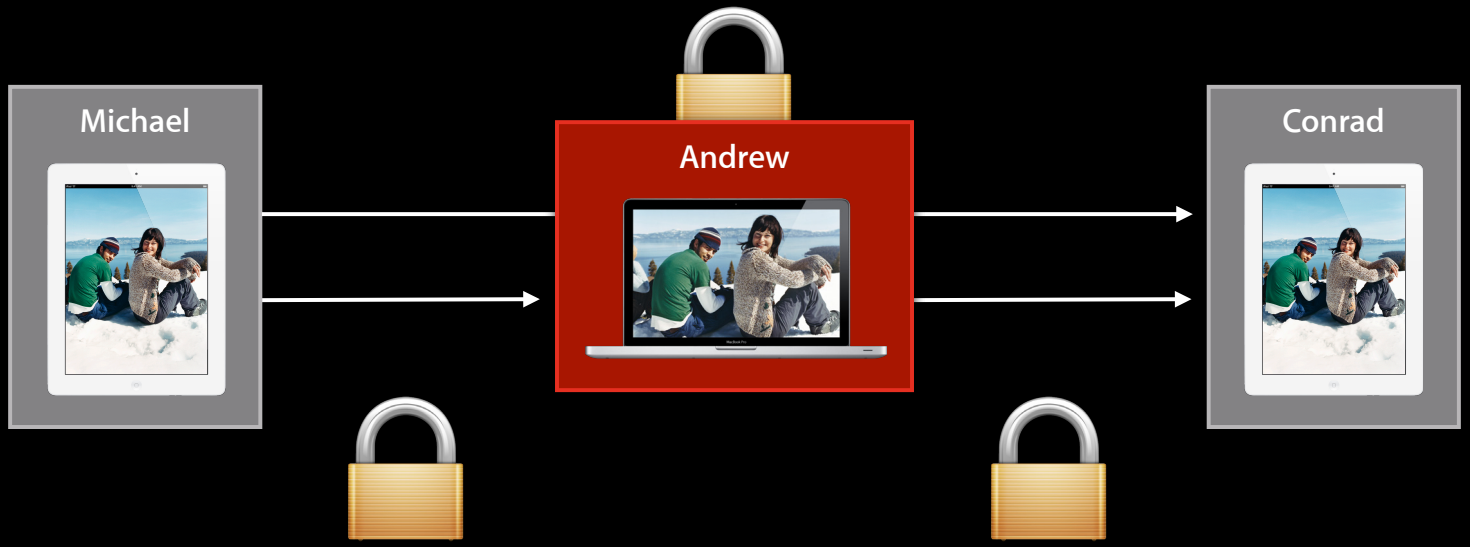
- No certificates, no problem

```
SSLCipherSuite cipher = TLS_DH_anon_WITH_AES_256_CBC_SHA;
SSLSetEnabledCiphers(ctx, &cipher, 1);
```

- Crypto alphabet soup
  - Diffie-Hellman key exchange
  - AES/CBC encryption with 256-bit keys
  - Secure Hash Algorithm (SHA1)

Man in the Middle

Encrypted, not secure



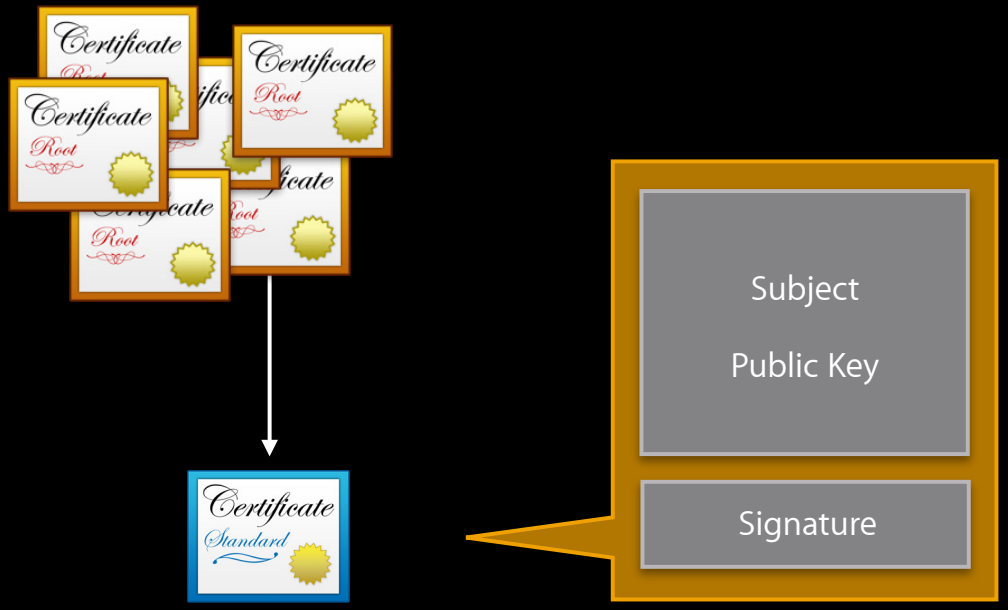
Certificate Authentication

How does it work?

- Using asymmetric keypair: Public and Private key
- Private key to create a signature over data
- Public key to verify signature

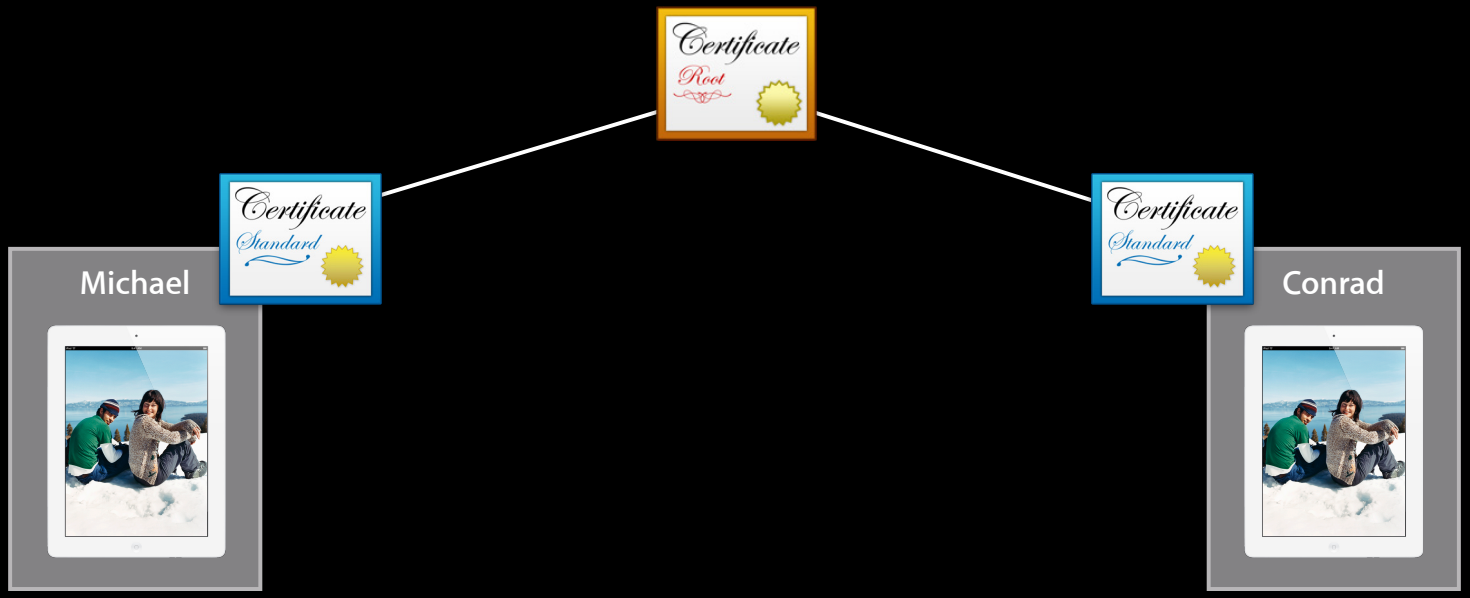
Certificates

What are they?



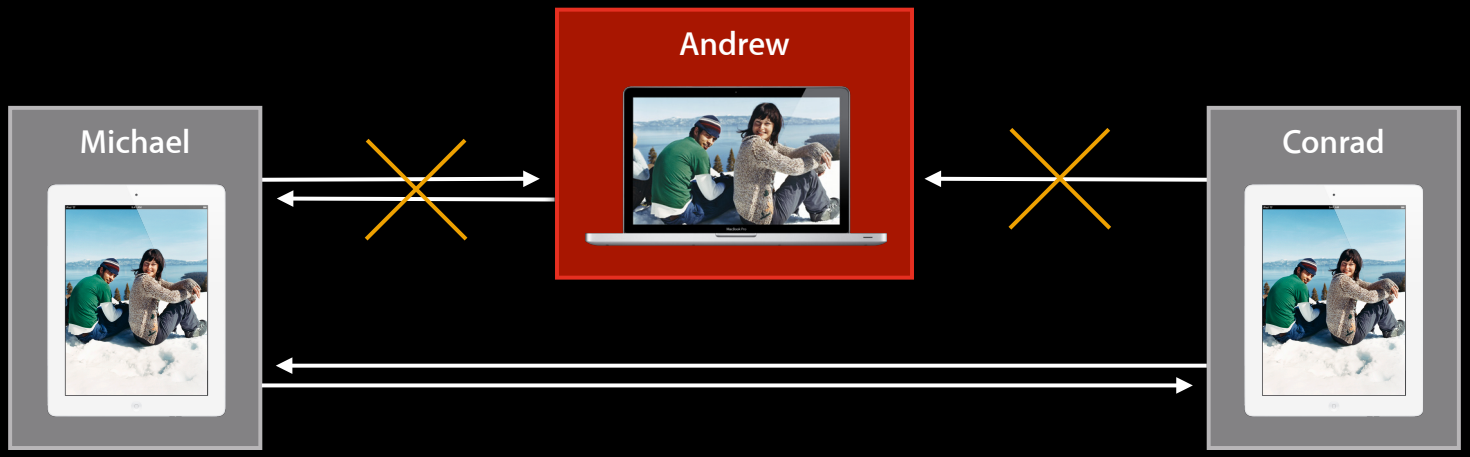
Certificates

Honest Abe's Lightly Used Bicycles and Cheap Certificates



Certificate Authentication

Server vs. client authentication



Certificate Authentication

Client authentication to peer

```
- (void)connection:(NSURLConnection *)c didReceiveAuthenticationChallenge:
(NSURLAuthenticationChallenge *)challenge {
    if ([[[challenge protectionSpace] authenticationMethod]
isEqualToString: NSURLAuthenticationMethodClientCertificate])
    {
        NSURLCredential * credential = [NSURLCredential
           credentialWithIdentity: identity
                     certificates: nil
                      persistence: NSURLCredentialPersistenceNone];
        [[challenge sender] useCredential: credential
               forAuthenticationChallenge: challenge];
    }
}
```

Certificate Authentication

Accept the challenge

```
- (BOOL)connection:(NSURLConnection *)connection
        canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)
                                                        protectionSpace
{
…
    if ([[protectionSpace authenticationMethod] isEqualToString:
                            NSURLAuthenticationMethodClientCertificate])
        return YES;
…
}
```

Demo

Certificate Authentication

Other things to consider

- Rule out "Man in the Middle"
- Trust the right certificates
- Limit Root Certificates (i.e., not Honest Abe's)

Authentication Using Certificates (Client)

Accept the challenge

```
- (BOOL)connection:(NSURLConnection *)connection
        canAuthenticateAgainstProtectionSpace:(NSURLProtectionSpace *)
                                                        protectionSpace
{
…
    else if ([[protectionSpace authenticationMethod] isEqualToString:
                                    NSURLAuthenticationMethodServerTrust])
        return YES;
…
}
```

Custom Trust Evaluation

Limit valid anchors as a client

```
if ([[[challenge protectionSpace] authenticationMethod] isEqualToString:
                                    NSURLAuthenticationMethodServerTrust]) {
    SecTrustRef trust = [[challenge protectionSpace] serverTrust];
    SecTrustSetAnchorCertificates(trust, oversharing_root);
    SecTrustEvaluate(trust, &trust_result);
    if (trust_result == kSecTrustResultUnspecified) {
        NSURLCredential * credential =
            [NSURLCredential credentialForTrust: trust];
        [[challenge sender] useCredential: credential
               forAuthenticationChallenge: challenge];
    }
}
```

Securing Network Connections

Summary

- HTTPS is easy to use from high-level API
- Requires some setup work, but provides simple experience
- Lot of options under the hood, so watch the sharp edges

Demo

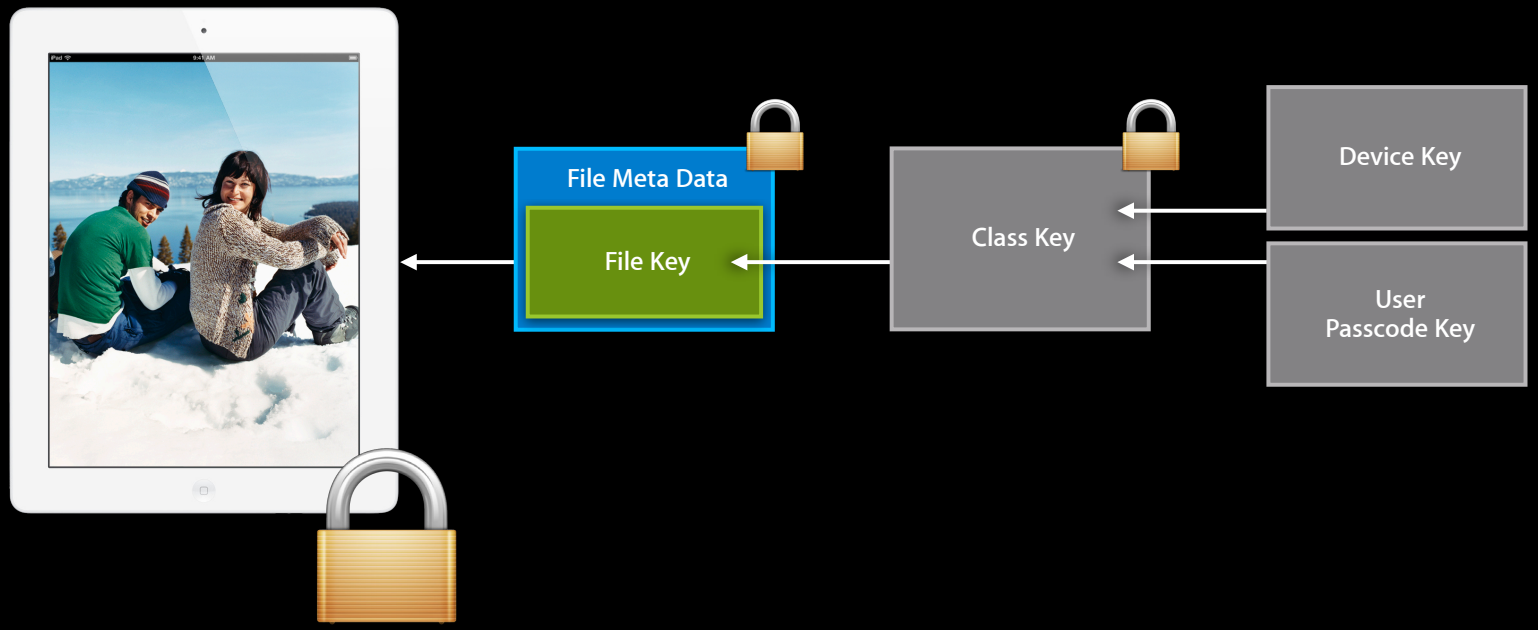
Protecting Data on the Device

Last line of defense

- Data Protection allows data to be tied to the user's passcode
- Provides protection in case a device is lost or stolen
  - Widely available hacking tools allow filesystem access

Data Protection Key Hierarchy

Protected file



APIs Offering Data Protection

NSFileManager	`NSFileProtectionKey`	`NSFileProtection…`
CoreData	`NSFileProtectionKey`	`NSFileProtection…`
NSData	`NSDataWritingOptions`	`NSDataWritingFileProtection…`
sqlite3	`sqlite3_open_v2 option`	`SQLITE_OPEN_FILEPROTECTION_…`
SecItem	`kSecAttrAccessible`	`kSecAttrAccessible…`

Data Protection for All Files

Foreground-only apps

- Add “DataProtectionClass” entitlement
- Use “NSFileProtectionComplete” as its value
- Profit!

Data Only Available When Unlocked

FileProtectionComplete

```
-(BOOL)writeImage:(UIImage *)image toPath:path error:(NSError **)error
{
    NSData *data = UIImageJPEGRepresentation(image, 1.0);
    return [data writeToFile:path options:
        NSDataWritingFileProtectionComplete error:error];
}
```

Data Only Available When Unlocked

Considerations

- Only as good as the passcode
- Cannot access when locked
  - Use `NSFileProtectionCompleteUnlessOpen`
  - Upgrade to `NSFileProtectionComplete` when unlocked

Data Dropbox

FileProtectionCompleteUnlessOpen

```
-(BOOL)writeImageWhileLocked:(UIImage *)image toPath:path
                       error:(NSError **)error
{
    NSData *data = UIImageJPEGRepresentation(image, 1.0);
    return [data writeToFile: path
                     options: NSDataWritingFileProtectionComplete
                       error: error]
        || [data writeToFile: path
                     options: NSDataWritingFileProtectionCompleteUnlessOpen
                       error: error];
}
```

Upgrade to FileProtectionComplete

```
-(void)upgradeImagesInDir:(NSString *)dir error:(NSError **)error {
    NSFileManager *fm = [NSFileManager defaultManager];
    NSDirectoryEnumerator *de = [fm enumeratorAtPath: dir];
    for (NSString *path in de) {
        NSDictionary *attrs = [de fileAttributes];
        if (![[attrs objectForKey: NSFileProtectionKey]
            isEqual: NSFileProtectionComplete]) {
            attrs = [NSDictionary dictionaryWithObject:
                NSFileProtectionComplete forKey: NSFileProtectionKey];
            [fm setAttributes: attrs ofItemAtPath: path error: error];
        }
    }
}
```

Background Read

…ProtectionCompleteUntilFirstUserAuthentication

- Also solves the problem of data access when unlocked
- Protects data from reboot until first unlock
  - Then not at all
  - Better than default of none against attacks that require a reboot

Background Readable Database

…ProtectionCompleteUntilFirstUserAuthentication

```
-(int)openDatabase(const char *filename, sqlite3 **pdb)
{
    return sqlite3_open_v2(filename, pdb,
        SQLITE_OPEN_FILEPROTECTION_COMPLETEUNTILFIRSTUSERAUTHENTICATION,
        NULL);
}
```

Demo

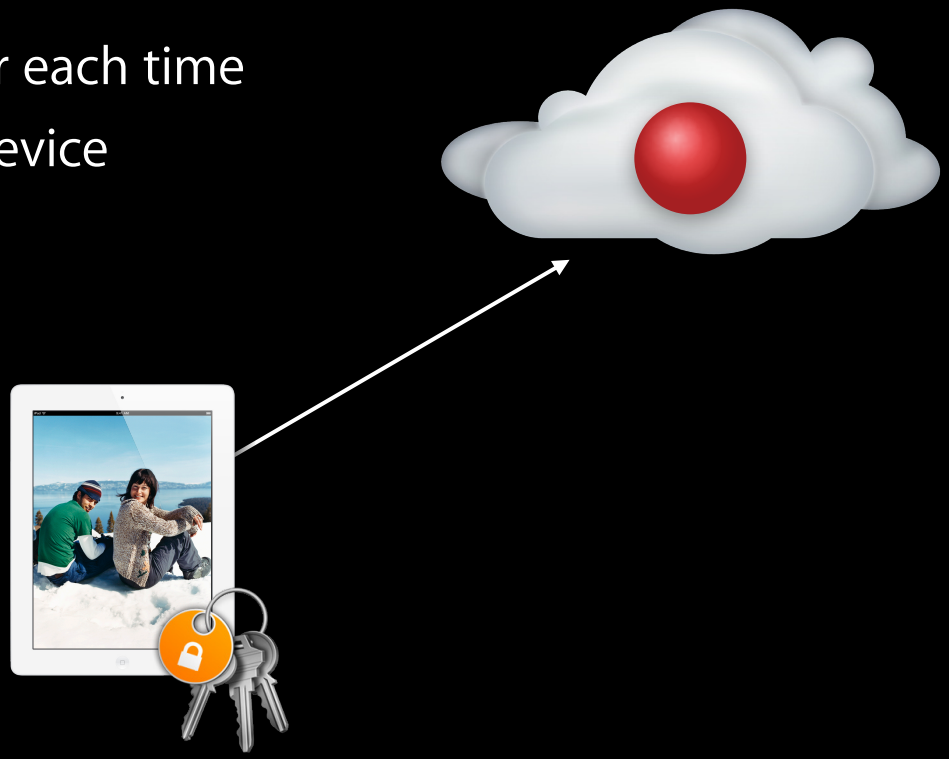
Keychain

Introduction

- What belongs in the keychain
- Keychain Items are protected just like files
  - Migratability can be controlled

Uploading to a Website

- Username and password
- Do not want to prompt the user each time
- Need to store securely on the device



Keychain vs. Data Protection Classes

Availability	NSFileProtection	kSecAttrAccessible
When unlocked	…Complete	…WhenUnlocked
While locked	…CompleteUnlessOpen	N/A
After first unlock	…CompleteUntilFirstUserAuthentication	…AfterFirstUnlock
Always	…None	…Always

Nonmigrating Keychain Classes

Availability	NSFileProtection	kSecAttrAccessible
When unlocked	…Complete	…WhenUnlockedThisDeviceOnly
While locked	…CompleteUnlessOpen	N/A
After first unlock	…CompleteUntilFirstUserAuthentication	…AfterFirstUnlockThisDeviceOnly
Always	…None	…AlwaysThisDeviceOnly

Keychain

Item lookup

```
- (NSMutableDictionary *) queryForAccount:(NSString *)account {
    return [NSMutableDictionary dictionaryWithObjectsAndKeys:
        kSecClassGenericPassword, kSecClass,
        @"Oversharing", kSecAttrService, account, kSecAttrAccount, nil];
}
- (NSData *) passwordForAccount:(NSString *)account found:(BOOL *)found {
    NSMutableDictionary *query = [self queryForAccount: account];
    [query setObject: kCFBooleanTrue forKey: kSecReturnData];
    NSData *data = NULL;
    OSStatus status = SecItemCopyMatching(query, &data);
    *found = status != errSecItemNotFound;
    return data;
}
```

Keychain

Item create

```
- (BOOL)setPassword: (NSData *)password forAccount:(NSString *)account {
    NSMutableDictionary *attrs = [self queryForAccount: account];
    [attrs setObject: password forKey: kSecValueData];
    [attrs setObject: kSecAttrAccessibleWhenUnlocked
        forKey: kSecAttrAccessible];
    OSStatus status = SecItemAdd(attrs, NULL);
    return status == noErr;
}
```

Keychain

Item update

```
- (BOOL)updatePassword: (NSData *)password forAccount:(NSString *)account {
    NSMutableDictionary *query = [self queryForAccount: account];
    NSMutableDictionary *attrs = [NSMutableDictionary dictionary];
    [attrs setObject: password forKey: kSecValueData];
    [attrs setObject: kSecAttrAccessibleWhenUnlocked
        forKey: kSecAttrAccessible];
    OSStatus status = SecItemUpdate(query, attrs);
    return status == noErr;
}
```

Keychain

High-level Keychain usage sample

```
-(BOOL) login: (NSString *) account {
    BOOL found = NO;
    NSData *pw = [self passwordForAccount: account found: &found];
    if ([self login: account password: pw]) return YES;
    pw = [self queryUserForPassword];
    if ([self login: account password: pw]) {
        if (found) [self updatePassword: pw forAccount: account];
        else [self setPassword: pw forAccount: account];
        return YES;
    }
    return NO;
}
```

Summary

- Protect your customers' data
  - Store secrets in the Keychain
  - Protect files with the best possible Data Protection class
  - Encrypt and authenticate network traffic

More Information

Keychain Services Reference

Certificate, Key, and Trust Services Reference

NSFileManager Class Reference

NSData Class Reference

CoreData Framework Reference

CFNetwork Framework Reference

Secure Transport Reference

Labs

Security Lab	Core OS Lab B Thursday 11:30AM
Security Lab	Core OS Lab B Friday 11:30AM


WWDC2011