Get Security and Privacy Right

Rob Napier

robnapier.net/ren2014

TODAY'S TOPICS

- Encrypting Network Traffic
- Data Protection
- Protecting Secrets
- Handling Passwords
- Correct AES Encryption

robnapier.net/ren2014

ENCRYPT YOUR TRAFFIC

HTTPS

- Payload Encryption
- URL Encryption
- Cookie Encryption
- Server Authentication
- Session Hijack Prevention
- Replay Attack Prevention

COMMERCIAL CERTS

- Sure, they're fine… but…
- Self-signed is better

A LOT OF TRUST

You Expect...

- Verisign
- Network Solutions
- Thawte
- RSA
- Digital Signature Trust

But Also...

- AOL, Cisco, Apple, ...
- US, Japan, Taiwan, ...
- Camerfirma, Dhimyotis, Echoworx, QuoVadis, Sertifitseerimiskeskus, Starfield, Vaestorekisterikeskus, ...

http://support.apple.com/kb/ht5012

$$T = Trust\ required$$

$$\forall T > 0 : T_{Self} + T_{Other} > T_{Self}$$

Don't Argue With Math

Self Signed Certificate

Certificate Pinning

```
#import "RNPinnedCertValidator.h"

- (void)connection:(NSURLConnection *)connection
willSendRequestForAuthenticationChallenge:(NSURLAuthenticationChallenge *)challenge {

  RNPinnedCertValidator *validator = [[RNPinnedCertValidator alloc]
                                          initWithCertificatePath:kPathToCerFile];
  [validator validateChallenge:challenge];
}
```

https://github.com/rnapier/RNPinnedCertValidator

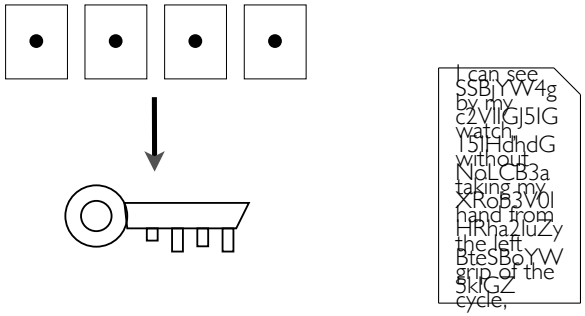
ENCRYPT YOUR TRAFFIC

- Use HTTPS for all traffic
- Pin your certs

https://github.com/rnapier/RNPinnedCertValidator

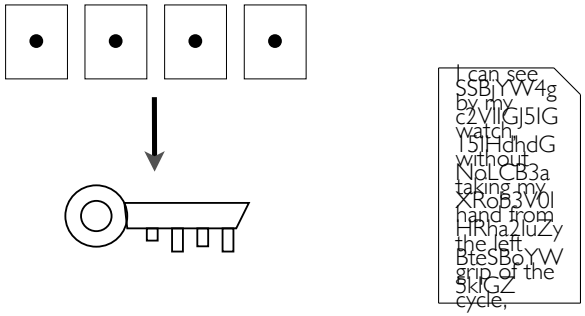
DATA PROTECTION

DATA PROTECTION (SIMPLIFIED)



`NSFileProtectionComplete`

DATA PROTECTION (SIMPLIFIED)

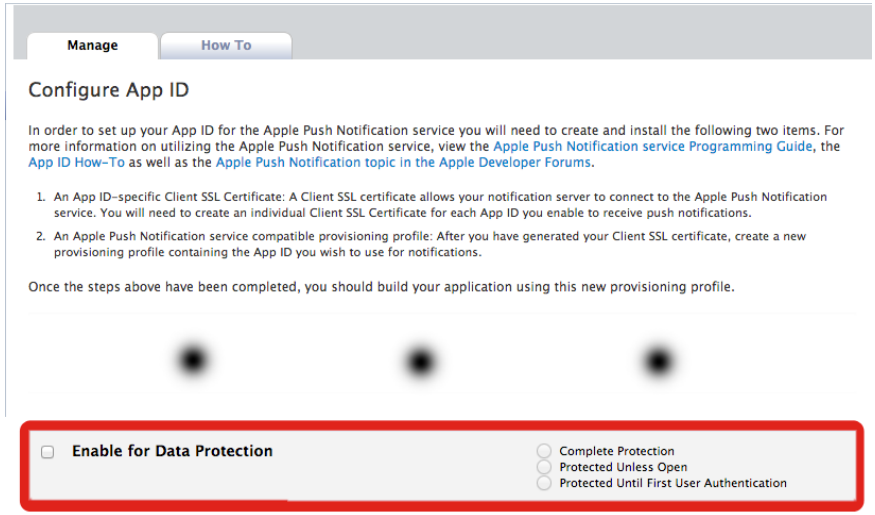


`NSFileProtectionComplete`

PROTECTION LEVELS

- Complete
- Complete Unless Open
- Complete Until First User Authentication

HOW EASY?


Manage
How To
Configure App ID
In order to set up your App ID for the Apple Push Notification service you will need to create and install the following two items. For more information on utilizing the Apple Push Notification service, view the Apple Push Notification service Programming Guide, the App ID How-To as well as the Apple Push Notification topic in the Apple Developer Forums.
1. An App ID-specific Client SSL Certificate: A Client SSL certificate allows your notification server to connect to the Apple Push Notification service. You will need to create an individual Client SSL Certificate for each App ID you enable to receive push notifications.
2. An Apple Push Notification service compatible provisioning profile: After you have generated your Client SSL certificate, create a new provisioning profile containing the App ID you wish to use for notifications.
Once the steps above have been completed, you should build your application using this new provisioning profile.
Enable for Data Protection
Complete Protection
Protected Unless Open
Protected Until First User Authentication

DATA PROTECTION IN CODE

```
[data writeToFile:path
          options:NSDataWritingFileProtectionComplete
            error:&error];
```

```
NSFileManager *fm = [NSFileManager defaultManager];
[fm setAttributes:@{
                    NSFileProtectionKey : NSFileProtectionComplete
                  }
     ofItemAtPath:path
            error:&error];
```

See **CompleteUnlessOpen** and **FileProtection** projects for examples

UIApplicationDelegate Methods

```
- (void)applicationProtectedDataWillBecomeUnavailable:(UIApplication *)application;
- (void)applicationProtectedDataDidBecomeAvailable:(UIApplication *)application;
```

UIApplication Notifications

```
UIKIT_EXTERN NSString *const UIApplicationProtectedDataWillBecomeUnavailable;
UIKIT_EXTERN NSString *const UIApplicationProtectedDataDidBecomeAvailable;
```

Note the missing "Notification"
rdar://13387084

UIApplication Methods

```
@property(nonatomic,readonly,getter=isProtectedDataAvailable) BOOL protectedDataAvailable;
```

https://www.apple.com/la/iphone/business/docs/iOS_Security_May12.pdf

https://www.apple.com/la/iphone/business/docs/iOS_Security_May12.pdf

DATA PROTECTION

- Turn it on automatically in your App ID
- Use Complete by default
- For background file access, try to use CompleteUnlessOpen
- Upgrade to Complete as soon as you can

PROTECTING SECRETS WITH KEYCHAIN

WHY KEYCHAIN?

- Automatically handles encryption
- Automatically handles backups/iCloud
- Incredibly persistent
- Sharing across applications

THE THING ABOUT KEYCHAIN…

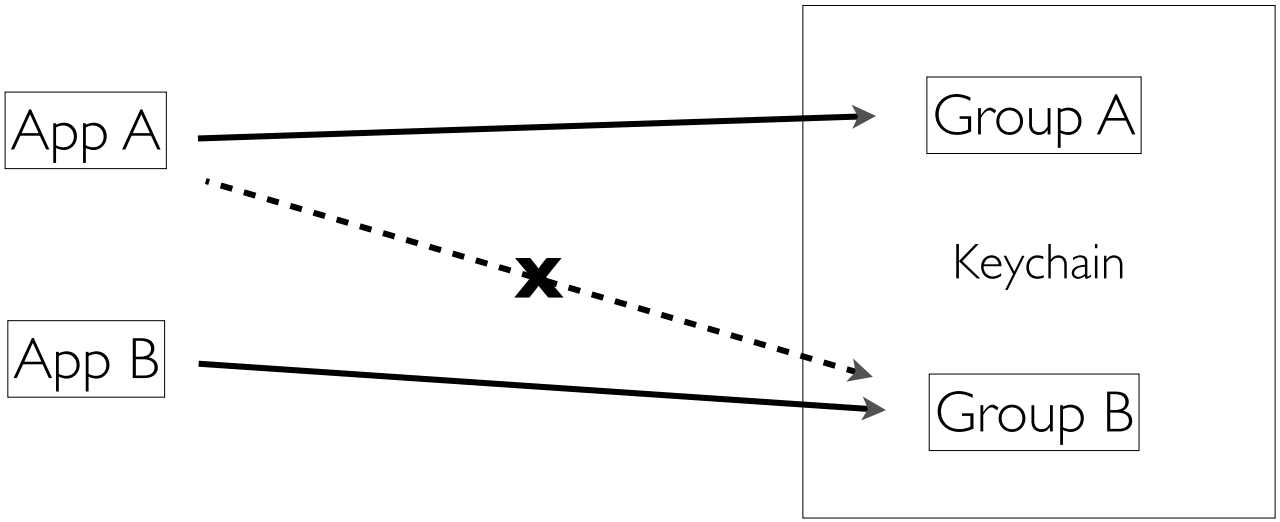
- Generally the best tool for the job, but…
  - A pain to use
  - Complicated
  - Slow

WRAPPERS

SGKeychain (https://github.com/secondgear/SGKeychain)

- Treat whole credential as an atomic unit
- Support access groups

ACCESS GROUPS

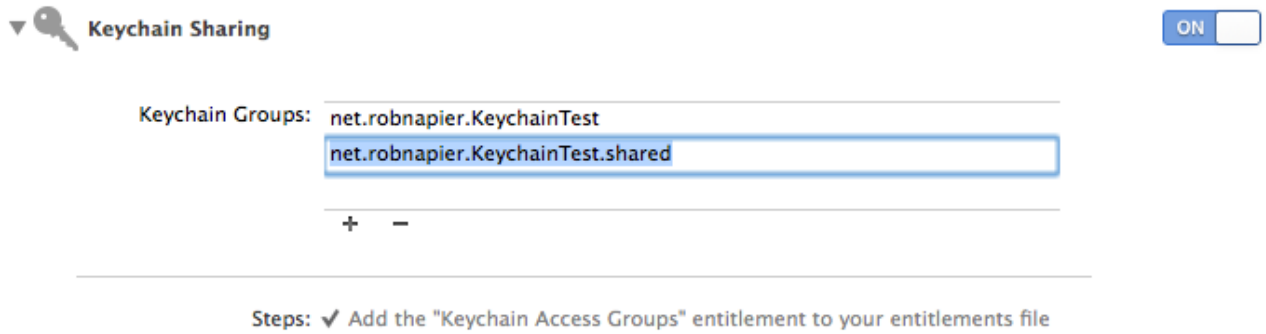


ACCESS GROUP FORMAT

<app-ID>.<reverse-DNS>.<identifier>

E9G2DXXXXX.net.robnapier.shared

ENTITLEMENTS



KeychainTest 〉 KeychainTest 〉 KeychainTest.entitlements 〉 No Selection

Key	Type	Value
▼ Entitlements File	Dictionary	(1 item)
▼ Keychain Access Groups	Array	(2 items)
Item 0	String	$(AppIdentifierPrefix)net.robnapier.KeychainTest
Item 1	String	$(AppIdentifierPrefix)net.robnapier.KeychainTest.shared

EXPLICIT ACCESS GROUPS

- If you're not explicit, it may work, but it may create duplicates
- I recommend requesting explicit access groups

```
NSString *accessGroup = @"...";
[SGKeychain setPassword:password
               username:username
            serviceName:service
            accessGroup:accessGroup
         updateExisting:YES
                  error:&error];
```

```
// Thanks to David H
// http://stackoverflow.com/q/11726672/97337
- (NSString *)applicationID {
  NSDictionary *query = @{ (__bridge id)kSecClass : (__bridge id)kSecClassGenericPassword,
                           (__bridge id)kSecAttrAccount : @"bundleSeedIDQuery",
                           (__bridge id)kSecAttrService : @"",
                           (__bridge id)kSecReturnAttributes : (id)kCFBooleanTrue
                           };
  CFDictionaryRef result = nil;
  OSStatus status = SecItemCopyMatching((__bridge CFTypeRef)query,
                                        (CFTypeRef *)&result);
  if (status == errSecItemNotFound)
    status = SecItemAdd((__bridge CFTypeRef)query, (CFTypeRef *)&result);
  if (status != errSecSuccess)
    return nil;
  NSString *accessGroup = [(__bridge NSDictionary *)result
                           objectForKey:(__bridge id)kSecAttrAccessGroup];
  NSArray *components = [accessGroup componentsSeparatedByString:@"."];
  NSString *bundleSeedID = components[0];
  CFRelease(result);
  return bundleSeedID;
}
```

```
NSString *accessGroup = [NSString stringWithFormat:@"%@.%@",
                          [self applicationID], kSharedKeychain];
[SGKeychain setPassword:password
               username:username
            serviceName:service
            accessGroup:accessGroup
         updateExisting:YES
                  error:&error];
```

KEYCHAIN

- Use a wrapper such as SGKeychain
- Use explicit access groups when sharing

Handling Passwords

HASHING

Password		Hash
`S3kr3t!`	→	`d39ee8e54ac7...`

A Cryptographic Hash is:

- Collision-resistant
- Preimage-resistant

CHOOSE YOUR HASH

- SHA-2 – Best commonly available
  - Pretty widely supported
  - No-known attacks
  - Also called SHA-224, -256, -384, and -512
- SHA-1 – Acceptable for most uses
  - Widely supported
  - Has known attacks, but not easy attacks
- SHA-3 – Someday
  - Can be faster than SHA-2
  - Few implementations

WHAT WENT WRONG?

d39ee8e54ac7f65311676d0cb92ec248319f7d27

Passw0rd	2acf37c868c0dd80513a4efa9ab4b4444a4d5c94
MyPass	b97698a2b0bf77a3e31e089ac5d43e96a8c34132
S3kr3t!	**d39ee8e54ac7f65311676d0cb92ec248319f7d27**
...	...

SALTING

Site 1

S3kr3t! → d39ee8e54ac7f65311676d0cb92ec248319f7d27

Site 2

S3kr3t! → d39ee8e54ac7f65311676d0cb92ec248319f7d27

SALTING

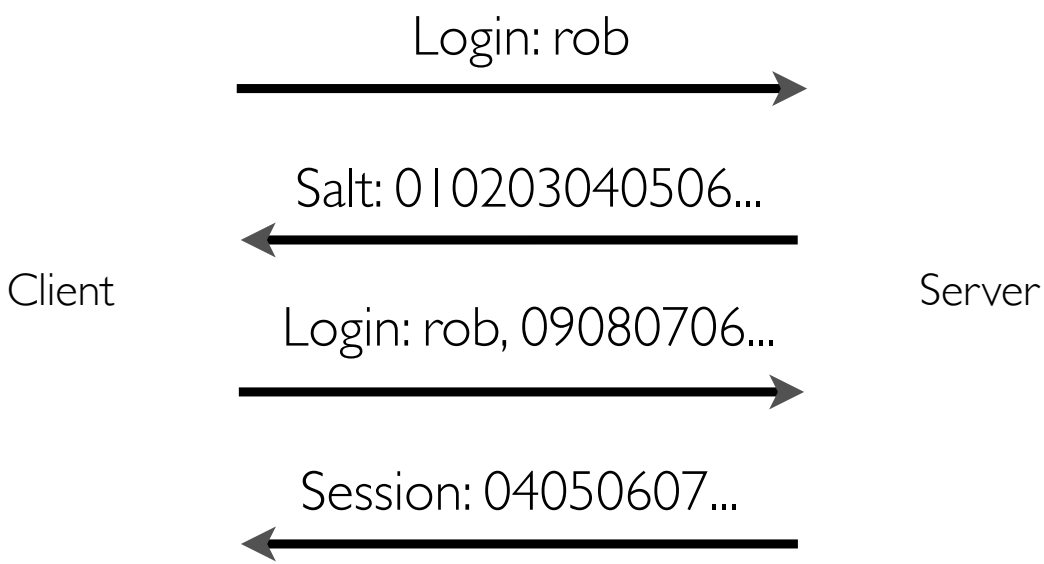
Site 1

XXX:S3kr3t! → 48fc6c1a82882c0084185c3e6f317d6cdabfbc88

Site 2

YYY:S3kr3t! → 7802cd6060f13349da21652e4bc8cd31e3058842

RANDOM SALT


Login: rob
Salt: 010203040506...
Client
Server
Login: rob, 09080706...
Session: 04050607...

DETERMINISTIC SALT

Prefix + *userid*

↓

com.example.MyGreatSite:robnapier@gmail.com

STRETCHING

- Real passwords are easy to guess
- To protect against that, make guessing expensive

TIME TO CRACK

	Guesses per second	Crack 8-char password
Native	1 billion	2 months
+80ms/guess	12.5	15 million years

PBKDF2

```
#import "RNCryptor.h"

- (NSData *)hashedPasswordForUsername:(NSString *)username password:(NSString *)password {

  // This is not a secret, but it should be unique to you
  // Random would be more secure, but is a headache if you want to login from multiple devices.
  static NSString *kHashPrefix = @"com.example.MyGreatApp";

  NSData *salt = [[NSString stringWithFormat:@"%@:%@", kHashPrefix, username]
                  dataUsingEncoding:NSUTF8StringEncoding];

  return [RNCryptor keyForPassword:password
                              salt:salt
                          settings:kRNCryptorAES256Settings.keySettings];
}
```

https://github.com/RNCryptor/RNCryptor

STORE A HASH

- Before storing the key in the database, hash it one more time with SHA-2

CONSISTENT-TIME CHECKS

```
@implementation NSData (RNSecureCompare)

- (BOOL)rnsc_isEqualInConsistentTime:(NSData *)otherData {
  // The point of this routine is XOR the bytes of each data and accumulate the results with OR.
  // If any bytes are different, then the OR will accumulate some non-0 value.
  uint8_t result = otherData.length - self.length;  // Start with 0 (equal) only if our lengths are equal

  const uint8_t *myBytes = [self bytes];
  const NSUInteger myLength = [self length];
  const uint8_t *otherBytes = [otherData bytes];
  const NSUInteger otherLength = [otherData length];

  for (NSUInteger i = 0; i < otherLength; ++i) {
    // Use mod to wrap around ourselves if they are longer than we are.
    // Remember, we already broke equality if our lengths are different.
    result |= myBytes[i % myLength] ^ otherBytes[i];
  }

  return result == 0;
}
@end
```

https://github.com/rnapier/NSData-RNSecureCompare

GOOD PASSWORD HANDLING

- Hash to hide the password
- Salt to make your hashes unique
- Stretch to make guessing slow
- Hash once more before storing
- Use consistent-time comparisons

CORRECT AES ENCRYPTION

USE MY LIBRARY

https://github.com/RNCryptor

USING RNCRYPTOR

```
#import "RNEncryptor.h"
NSData *encryptedData = [RNEncryptor encryptData:data
                                withSettings:kRNCryptorAES256Settings
                                    password:aPassword
                                       error:&error];
```

```
#import "RNDecryptor.h"
NSData *decryptedData = [RNDecryptor decryptData:encryptedData
                                withPassword:aPassword
                                       error:&error];
```

- iOS / OS X
- C++
- C#
- Java
- JavaScript (soon)
- PHP
- Python
- Ruby

WHAT IS CORRECT AES?

Hold that thought…

P1	P2	P3	P4
P5	P6	P7	P8
P9	P10	P11	P12
P13	P14	P15	P16

Key

Encrypt

C1	C2	C3	C4
C5	C6	C7	C8
C9	C10	C11	C12
C13	C14	C15	C16

P1	P2	P3	P4
P5	P6	P7	P8
P9	P10	P11	P12
P13	P14	P15	P16

Key

Decrypt

C1	C2	C3	C4
C5	C6	C7	C8
C9	C10	C11	C12
C13	C14	C15	C16

THE HELPERS

- Key Generation
- Block Cipher Modes
- Authentication

INCORRECT KEY GENERATION

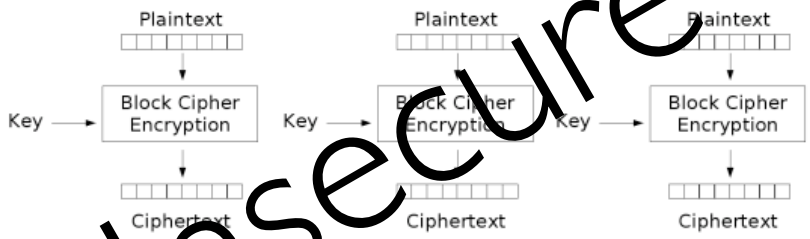
```
// This is broken
NSString *password = @"P4ssW0rd!";

char key[kCCKeySizeAES256+1];
bzero(key, sizeof(key));

[key getCString:keyPtr maxLength:sizeof(keyPtr) encoding:NSUTF8StringEncoding];
// This is broken
```

- Truncates long passwords
- Uses only a tiny part of the key space
  - Best case is ~ 0.00001% of a 128-bit key.

Use a PBKDF
(scrypt, bcrypt, PBKDF2)

INITIALIZATION VECTOR

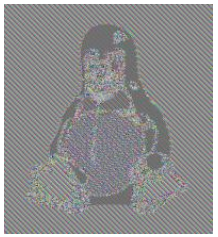
And Modes of Operation



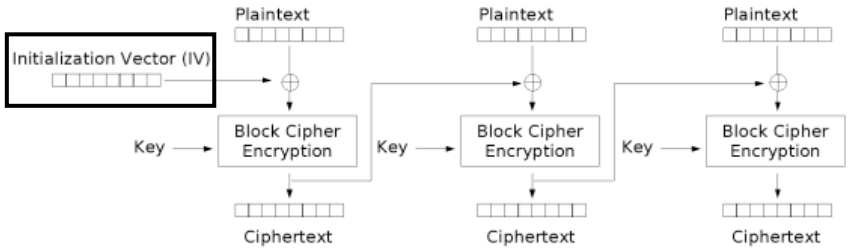
Electronic Codebook (ECB) mode encryption

Insecure

ECB



Source image by Larry Ewing <lewing@isc.tamu.edu> and The GIMP



Cipher Block Chaining (CBC) mode encryption

SO MUCH CONFUSION FROM ONE COMMENT

```
CCCryptorStatus CCCryptorCreate(
    CCOperation op,             /* kCCEncrypt, etc. */
    CCAlgorithm alg,            /* kCCAlgorithmDES, etc. */
    CCOptions options,          /* kCCOptionPKCS7Padding, etc. */
    const void *key,            /* raw key material */
    size_t keyLength,
    const void *iv,             /* optional initialization vector */
    CCCryptorRef *cryptorRef)  /* RETURNED */
__OSX_AVAILABLE_STARTING(__MAC_10_4, __IPHONE_2_0);
```

Use an unpredictable IV, not NULL.

UNAUTHENTICATED ENCRYPTION

Amt:$100.To:**Bob**.From:Alice.Seq:PQ123.Comment:Here's the money I owe you.

↓

Alice encrypts

↓

Eve modifies message

↓

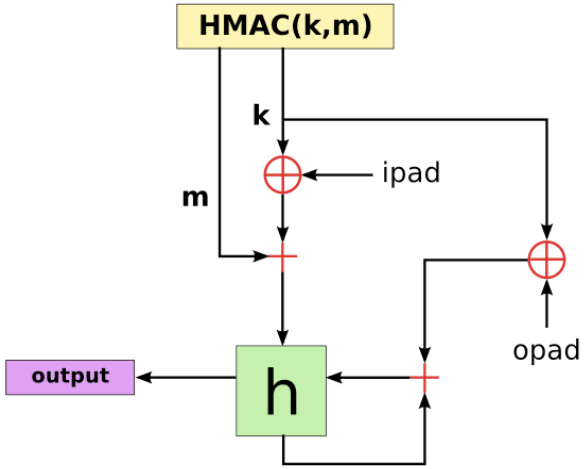
Bank decrypts

↓

Amt:$100.To:**Eve**.From:Alice.Seq:PQ123.Comment:Here's the money I owe you.

```
# The string Eve would like to inject and the location.
newMsg = "Eve"
newMsgLoc = 12

# Eve has access to cipher and to iv. She calculates a new iv that will modify
# how the first block is decrypted. For each byte she wants to replace, she
# calculates (original_iv ^ original_msg ^ new_msg) where ^ is xor.
new_iv = list(iv)
for index in range(newMsgLoc, newMsgLoc + len(newMsg)):
  new_iv[index] = chr(ord(iv[index]) ^ ord(msg[index]) ^ ord(newMsg[index - newMsgLoc]))
new_iv = ''.join(new_iv)
```

See modaes.py for full example



HASH BASED MESSAGE AUTHENTICATION CODE

COMPUTING HMAC

```
CCHmac(kCCHmacAlgSHA512,   // algorithm
       [hmacKey bytes],    // key
       [hmacKey length],   // keyLength
       [message bytes],    // data
       [message length],   // dataLength
       [hmac mutableBytes] // macOut
       );
```

message must be whole message

ENCRYPTION PITFALLS

- Poor KDF choice
- Truncating multi-byte passwords
- Insufficiently random salt
- Key truncation
- Poor block cipher mode choice
- Predictable IV
- No HMAC
- Failure to HMAC entire message
- Poor cipher choice
- Key/IV reuse
- Failure to validate padding
- Failure to validate HMAC
- Length-extension attacks
- Timing attacks
- Side-channel attacks
- Ciphertext truncation attacks

ENCRYPTION PITFALLS

- Poor KDF choice
- Truncating multi-byte passwords
- Insufficiently random salt
- Key truncation
- Poor block cipher mode choice
- Predictable IV
- No HMAC
- Failure to HMAC entire message
- Poor cipher choice
- Key/IV reuse
- Failure to validate padding
- Failure to validate HMAC
- Length-extension attacks
- Timing attacks
- Side-channel attacks
- Ciphertext truncation attacks

DON'T BUILD YOUR OWN AES FORMAT

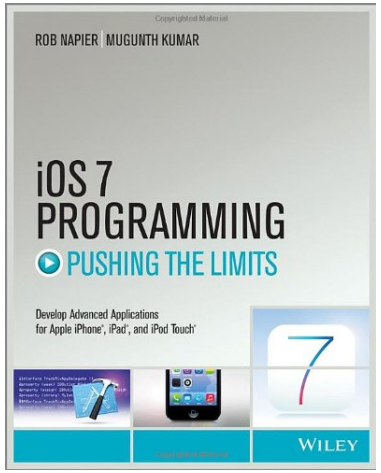
WHAT SHOULD YOU DO?

- RNCryptor - rncryptor.github.io
- AES Crypt - aescrypt.com
- Hire a security specialist or become one

PRACTICAL SECURITY

- Encrypt your traffic with SSL
- Pin and verify your certs (RNPinnedCertValidator)
- Encrypt your files with ProtectionComplete
- Use SGKeychain for storing passwords
- Salt and stretch your passwords
- Use AES correctly with RNCryptor

robnapier.net/ren2014
robnapier@gmail.com
@cocoaphony
αrobnapier



iOS 7 Programming Pushing The Limits
Chapter 14

iosptl.com