Chapter 57: PBKDF2 Key Derivation

Section 57.1: Password Based Key Derivation 2 (Swift 3)

Password Based Key Derivation can be used both for deriving an encryption key from password text and saving a password for authentication purposes.

There are several hash algorithms that can be used including SHA1, SHA256, SHA512 which are provided by this example code.

The rounds parameter is used to make the calculation slow so that an attacker will have to spend substantial time on each attempt. Typical delay values fall in the 100ms to 500ms, shorter values can be used if there is unacceptable performance.

This example requires Common Crypto
It is necessary to have a bridging header to the project:
`#import <CommonCrypto/CommonCrypto.h>`
Add the `Security.framework` to the project.

Parameters:

```
password     password String
salt         salt Data
keyByteCount number of key bytes to generate
rounds       Iteration rounds

returns      Derived key


func pbkdf2SHA1(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA256(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2SHA512(password: String, salt: Data, keyByteCount: Int, rounds: Int) -> Data? {
    return pbkdf2(hash:CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyByteCount:keyByteCount, rounds:rounds)
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: Data, keyByteCount: Int, rounds: Int)
-> Data? {
    let passwordData = password.data(using:String.Encoding.utf8)!
    var derivedKeyData = Data(repeating:0, count:keyByteCount)

    let derivationStatus = derivedKeyData.withUnsafeMutableBytes {derivedKeyBytes in
        salt.withUnsafeBytes { saltBytes in

            CCKeyDerivationPBKDF(
                CCPBKDFAlgorithm(kCCPBKDF2),
                password, passwordData.count,
                saltBytes, salt.count,
                hash,
                UInt32(rounds),
```

```
                derivedKeyBytes, derivedKeyData.count)
        }
    }
    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKeyData
}
```

Example usage:

```
let password     = "password"
//let salt       = "saltData".data(using: String.Encoding.utf8)!
let salt         = Data(bytes: [0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let keyByteCount = 16
let rounds       = 100000

let derivedKey = pbkdf2SHA1(password:password, salt:salt, keyByteCount:keyByteCount, rounds:rounds)
print("derivedKey (SHA1): \(derivedKey! as NSData)")
```

Example Output:

```
derivedKey (SHA1): <6b9d4fa3 0385d128 f6d196ee 3f1d6dbf>
```

Section 57.2: Password Based Key Derivation 2 (Swift 2.3)

See Swift 3 example for usage information and notes

```
func pbkdf2SHA1(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA1), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA256(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA256), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2SHA512(password: String, salt: [UInt8], keyCount: Int, rounds: Int) -> [UInt8]? {
    return pbkdf2(CCPBKDFAlgorithm(kCCPRFHmacAlgSHA512), password:password, salt:salt,
keyCount:keyCount, rounds:UInt32(rounds))
}

func pbkdf2(hash :CCPBKDFAlgorithm, password: String, salt: [UInt8], keyCount: Int, rounds:
UInt32!) -> [UInt8]! {
    let derivedKey   = [UInt8](count:keyCount, repeatedValue:0)
    let passwordData = password.dataUsingEncoding(NSUTF8StringEncoding)!

    let derivationStatus = CCKeyDerivationPBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        UnsafePointer<Int8>(passwordData.bytes), passwordData.length,
        UnsafePointer<UInt8>(salt), salt.count,
        CCPseudoRandomAlgorithm(hash),
        rounds,
        UnsafeMutablePointer<UInt8>(derivedKey),
        derivedKey.count)
```

```
    if (derivationStatus != 0) {
        print("Error: \(derivationStatus)")
        return nil;
    }

    return derivedKey
}
```

Example usage:

```
let password = "password"
// let salt  = [UInt8]("saltData".utf8)
let salt     = [UInt8]([0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let rounds   = 100_000
let keyCount = 16

let derivedKey = pbkdf2SHA1(password, salt:salt, keyCount:keyCount, rounds:rounds)
print("derivedKey (SHA1):   \(NSData(bytes:derivedKey!, length:derivedKey!.count))")
```

Example Output:

```
derivedKey (SHA1):   <6b9d4fa3 0385d128 f6d196ee 3f1d6dbf>
```

Section 57.3: Password Based Key Derivation Calibration (Swift 2.3)

See Swift 3 example for usage information and notes

```
func pbkdf2SHA1Calibrate(password:String, salt:[UInt8], msec:Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
        UInt32(msec));
    return actualRoundCount
}
```

Example usage:

```
let saltData       = [UInt8]([0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let passwordString = "password"
let delayMsec      = 100

let rounds = pbkdf2SHA1Calibrate(passwordString, salt:saltData, msec:delayMsec)
print("For \(delayMsec) msec delay, rounds: \(rounds)")
```

Example Output:

> For 100 msec delay, rounds: 94339

Section 57.4: Password Based Key Derivation Calibration

(Swift 3)

Determine the number of PRF rounds to use for a specific delay on the current platform.

Several parameters are defaulted to representative values that should not materially affect the round count.

```
password Sample password.
salt     Sample salt.
msec     Targeted duration we want to achieve for a key derivation.

returns  The number of iterations to use for the desired processing time.


func pbkdf2SHA1Calibrate(password: String, salt: Data, msec: Int) -> UInt32 {
    let actualRoundCount: UInt32 = CCCalibratePBKDF(
        CCPBKDFAlgorithm(kCCPBKDF2),
        password.utf8.count,
        salt.count,
        CCPseudoRandomAlgorithm(kCCPRFHmacAlgSHA1),
        kCCKeySizeAES256,
        UInt32(msec));
    return actualRoundCount
}
```

Example usage:

```
let saltData       = Data(bytes: [0x73, 0x61, 0x6c, 0x74, 0x44, 0x61, 0x74, 0x61])
let passwordString = "password"
let delayMsec      = 100

let rounds = pbkdf2SHA1Calibrate(password:passwordString, salt:saltData, msec:delayMsec)
print("For \(delayMsec) msec delay, rounds: \(rounds)")
```

Example Output:

```
For 100 msec delay, rounds: 93457
```