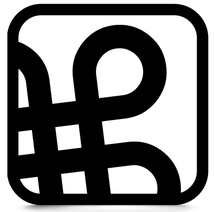


# Macoun

# Der Schlüssel zum Glück

Klaus Rodewig

# KMR

- Mobiliar im Slack-Channel cocoaheadsde.slack.com
- Einladungen zum Slack-Channel bei @sebastianbachmann

# Überrasch' Deine Eltern
# - lies ein Buch!


mov @dasdom,@cocoanehead


mov Swift,Objective-C

# Onprangering

- keine Sprecherin auf der Macoun 2015
- kein Vortrag von Amin
- zu viele Vorträge alter Säcke
- mindestens ein alter Sack hält zwei Vorträge
- Dr. Autolayout Hauser fehlt

# Waff?

- The same procedure as every year
- Fakten, Fakten, Fakten
- Dieses und jenes
- Verschiedenes

Das
*KMR Institut für cloudbasiertes Cyber Management*
präsentiert eine …

# Kurze Umfrage

Wer hat eine oder mehrere Apps im Store oder im Unternehmen?

# Wessen App verarbeitet personenbezogene oder sensible Daten?

# Was sind personenbezogene oder sensible Daten?

# Wer verwendet die Keychain?

Wer verwendet für den Zugriff auf die Keychain eine 3rd Party Lib?

Welche App speichert personenbezogene oder sensible Daten lokal?

# Wer verwendet NSFileProtection?

# Wer verwendet CommonCrypto?

# Welche App kommuniziert per HTTPS mit einem Backend?

Wer verwendet für HTTPS eine 3rd Pary Lib (e.g. AFNetworking)?

# Welche App macht Certificate Pinning?

# Was ist Certificate Pinning?

# Beispiel*

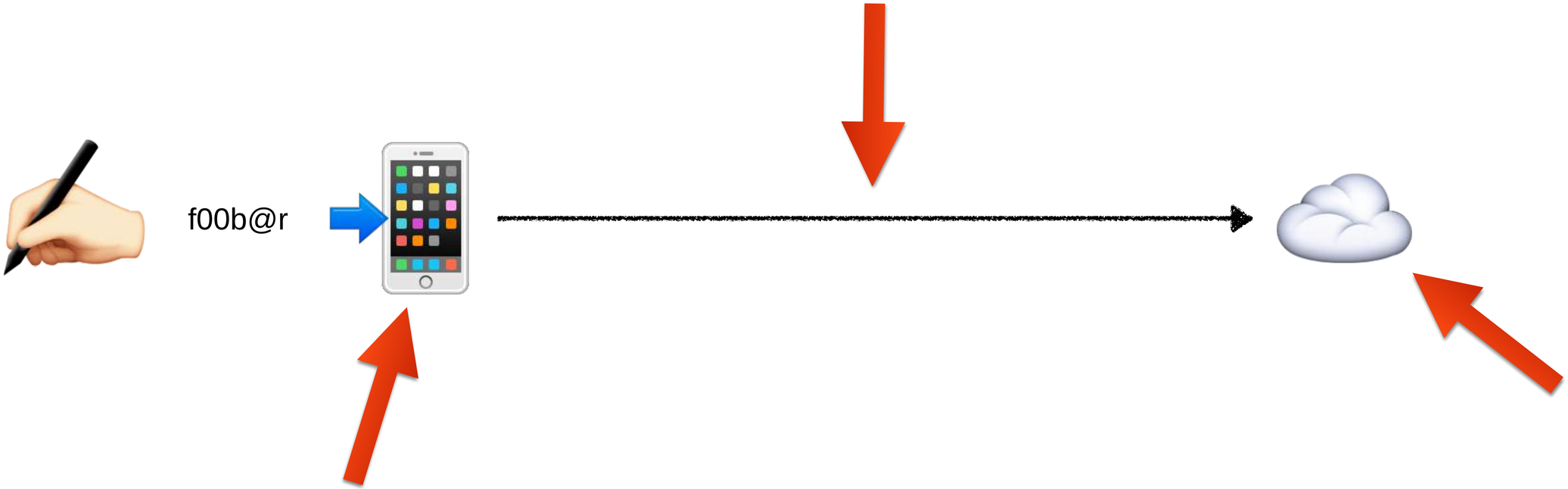
- Client-Server-App für iOS
- Twittbook für Unternehmen
- Daten: Zugangsdaten, Personaldaten, Interna, Kundendaten, Fotos vom Betriebsausflug, Trivialitäten

* von September 2015

# Zugangsdaten

- Benutzername
- Passwort

# Passwort



# Twittbook

- Passwort im Klartext in lokaler Datei
- Passwort im Klartext an Server
- Passwort im Klartext auf Server

# ~~Was~~ Waff nun?

- Server benötigt **eindeutiges** Identifikationsmerkmal
- Kein Ermitteln des Passwortes
- kollisionsresistene Einwegfunktion (aka Hash)

# Hash aus Amsterdam

- SHA-1 und MD5 weit verbreitet, aber grundsätzlich b0rken
- “Kryptographische Verfahren: Empfehlungen und Schlüssellängen / BSI TR-02102-1”, Bundesamt für Sicherheit in der Informationstechnik
- für Einsatz nach 2015: SHA-256 oder höher

# Servicefolie: Swift-Bashing

- CommonCrypto
- $ man CC_SHA256


```
extern unsigned char * CC_SHA256(const void *data, CC_LONG len, unsigned char *md);
```

- != Playground

# Passwort als Hash


- keine direkte Nutzung als Passwort mehr möglich
- Kein Missbrauch auf anderen Plattformen

# Layer-8-Problem

- einfache Passwörter
- Rainbowtables: vorausberechnete Hash-Tabellen*
- www.freerainbowtables.com
  - 9741 GB
- Im “Dark Web” noch weitere Quellen
- Zurückrechnen eines Passwortes aus einem Hash

* stark vereinfacht

# Salt

- zufällige Zeichenfolge
- Verknüpfung mit Passwort
- keine Vorausberechnung mehr möglich
  - MD5 rainbow tables (3889 GB)
  - Länge 1-9
  - alphanummerisch, Sonderzeichen

# Umgang mit Salt

- muss **nicht** geheim sein
- sollte für jeden User anders sein
- muss persistent sein
  - z.B. Keychain
  - Backend

# Keychain

| **Verfügbar** | **Protection class** |
|---|---|
| Immer | `kSecAttrAccessibleAlways` |
| Immer, nur dieses Gerät | `kSecAttrAccessibleAlwaysThisDeviceOnly` |
| Nach **erster** Passcode-Eingabe | `kSecAttrAccessibleAfterFirstUnlock` |
| Nach **erster** Passcode-Eingabe, nur **dieses** Gerät | `kSecAttrAccessibleAfterFirstUnlockThisDeviceOnly` |
| Nur, wenn Gerät entsperrt ist | `kSecAttrAccessibleWhenUnlocked` |
| Nur, wenn **dieses** Gerät entsperrt ist | `kSecAttrAccessibleWhenUnlockedThisDeviceOnly` |
| Nur, wenn **dieses** Gerät entsperrt ist und ein Passcode gesetzt ist. | `kSecAttrAccessibleWhenPasscodeSetThisDeviceOnly` |

# Keychain-API

```
OSStatus SecItemAdd ( CFDictionaryRef attributes, CFTypeRef _Nullable *result );

OSStatus SecItemUpdate ( CFDictionaryRef query, CFDictionaryRef
attributesToUpdate );

OSStatus SecItemDelete ( CFDictionaryRef query );

OSStatus SecItemCopyMatching ( CFDictionaryRef query, CFTypeRef _Nullable
*result );
```

# SecItemAdd


Dienst-Beschreibung
Acount (z.B. Username)
Keychain-Typ
Beschreibung für User
NSMutableDictionary *theWriteDict = [NSMutableDictionary dictionary];
[theWriteDict setObject:(__bridge id)kSecClassGenericPassword forKey:(__bridge id)kSecClass];
[theWriteDict setObject:inService forKey:(__bridge id)kSecAttrService];
[theWriteDict setObject:inLabel forKey:(__bridge id)kSecAttrLabel];
[theWriteDict setObject:inAccount forKey:(__bridge id)kSecAttrAccount];CFTypeRef
theProtectionClass = inProtectionClass ? inProtectionClass : kSecDefaultProtectionClass;
[theWriteDict setObject:(__bridge id)theProtectionClass forKey:(__bridge id)kSecAttrAccessible];
[theWriteDict setObject:inSecret forKey:(__bridge id)kSecValueData];
if(inAccessGroup != nil){
[theWriteDict setObject:inAccessGroup forKey:(__bridge id)kSecAttrAccessGroup];
}
SecItemAdd((__bridge CFDictionaryRef)theWriteDict, NULL);
Geheimnis (z.B. Passwort)
übergreifender Zugriff
Protection Class

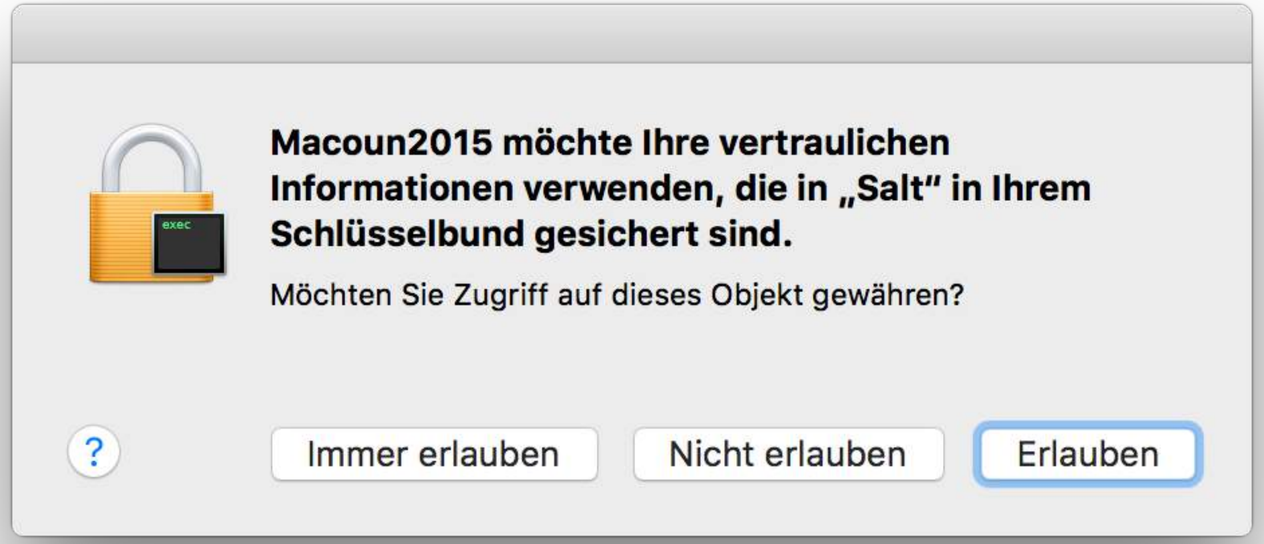
# errSecDuplicateItem

```
//errSecDuplicateItem = -25299,  /* The specified item already exists in the keychain. */


if(!(SecItemAdd((__bridge CFDictionaryRef)theWriteDict, NULL)){
        SecItemUpdate((__bridge CFDictionaryRef)theSearchDict,
        ((__bridge CFDictionaryRef)theUpdateDict));
}
```

# SecItemCopyMatching

```
NSMutableDictionary *theQueryDict = [NSMutableDictionary dictionary];
[theQueryDict setObject:(__bridge NSString *)kSecClassGenericPassword forKey:(__bridge NSString *)kSecClass];
[theQueryDict setObject:inAccount forKey:(__bridge id)kSecAttrAccount];
[theQueryDict setObject:inLabel forKey:(__bridge id)kSecAttrLabel];
[theQueryDict setObject:inService forKey:(__bridge id)kSecAttrService];
[theQueryDict setObject:(id)kCFBooleanTrue forKey:(__bridge_transfer id)kSecReturnData];

CFDataRef thePWData = nil;
OSStatus theStatus = SecItemCopyMatching((__bridge CFDictionaryRef)theQueryDict, (CFTypeRef*)&thePWData);
```

# Siehste


exec
Macoun2015 möchte Ihre vertraulichen Informationen verwenden, die in „Salt“ in Ihrem Schlüsselbund gesichert sind.
Möchten Sie Zugriff auf dieses Objekt gewähren?
?
Immer erlauben
Nicht erlauben
Erlauben

# App-übergreifender Zugriff

- Access Group beim Schreiben in Keychain
- Access Group in Entitlements

# Verschlüsselte Dateien

- NSFileProtection
- analog zu Protection Classes der Keychain
- Aktivierung über Attribut
- wirkt **nur** lokal

# Beispiel

```
[[NSFileManager defaultManager] createFileAtPath:[self filePath]
                                        contents:[@"foobar"
                                    dataUsingEncoding:NSUTF8StringEncoding]
                                          attributes:[NSDictionary
                                   dictionaryWithObject:NSFileProtectionComplete
                                                 forKey:NSFileProtectionKey]];
```

# Verschlüsselung im Eigenbau

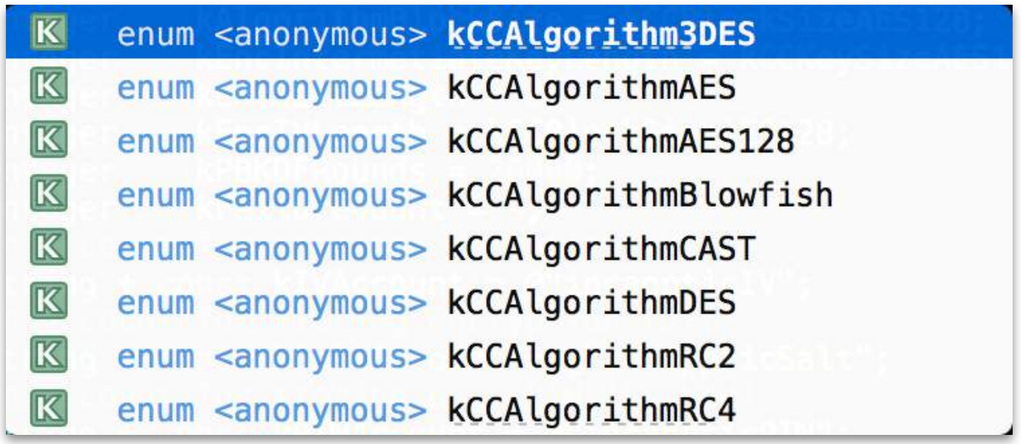
```
#include <CommonCrypto/CommonCryptor.h>

CCCrypt(CCOperation op, CCAlgorithm alg, CCOptions options, const void *key, size_t
keyLength, const void *iv, const void *dataIn, size_t dataInLength, void *dataOut, size_t
dataOutAvailable, size_t *dataOutMoved);
```

# CCCrypt


Algorithmus
Padding
Schlüssel
Länge des Schlüssels
Initialisierungsvektor
Klartext
Länge des Klartextes
Ciphertext
Länges des Ciphertextes
Länge der Daten
CCCryptorStatus theResult = CCCrypt(kCCEncrypt,
kCCAlgorithmAES128,
kCCOptionPKCS7Padding,
theEncryptionKey.bytes,
theEncryptionKey.length,
theIV.bytes,
inData.bytes,
inData.length,
theCipherText.mutableBytes,
theCipherText.length,
outLength);

# Algorithmus

- BSI TR 02102-1



# Padding

- Notwendig bei der Verwendung von Blockchiffren
- sonst: letzter (unvollständiger) Block leakt Klartext

# Initialisierungsvektor

- gleicher Schlüssel + gleiche Daten führen beim ersten Block zum gleichen Ciphertext
- Initialisierungsvektor fügt Zufall hinzu, um gleichen Klartext in unterschiedlichen Ciphertext zu überführen
- Wohin damit? An den Ciphertext kleben.

# PBKDF2 / BCRYPT

- Password-Based Key Derivation Function 2
- Iterative Anwendung von Hashfunktionen auf das Passwort
- erschwert Brute-Force-Angriffe auf den Schlüssel
- PBKDF2 ist nicht optimal, aber in iOS verfügbar
- besser: bcrypt oder scrypt

# TLS - Certificate Pinning

```
 if ([[[challenge protectionSpace] authenticationMethod]
isEqualToString:NSURLAuthenticationMethodServerTrust]) {
        SecTrustRef serverTrust = [[challenge protectionSpace] serverTrust];
        BOOL trustedCert = NO;
        NSData *theData = [NSData new];
        if(serverTrust != NULL) {
            CFIndex theCertCount = SecTrustGetCertificateCount(serverTrust);
            for(CFIndex theCertIndex = 0; theCertIndex < theCertCount; theCertIndex++)                {
                SecCertificateRef theCert =
                        SecTrustGetCertificateAtIndex(serverTrust, theCertIndex);
                theData = (__bridge_transfer NSData *)SecCertificateCopyData(theCert);
                if([kSecurityCloudHash caseInsensitiveCompare:[self sha1HexDigest:theData]]
                == NSOrderedSame){
                    trustedCert = YES;
                    break;
                } else {
                    trustedCert = NO;
                }
            }
```

# TLS - Certificate Pinning

- Secure by default
- TLS 1.2 mit Perfect Forward Secrecy ist jetzt Standard
- wirkt auf:
  - NSURLSession
  - NSURLConnection
  - CFURL
- aktiv, wenn App für iOS 9 gelinkt wird

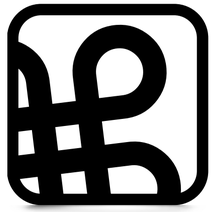
# Opt-out in Info.plist

| Key | Type |
| --- | --- |
| NSAppTransportSecurity | Dictionary |
| NSAllowsArbitraryLoads | Boolean |
| NSExceptionDomains | Dictionary |
| <domain-name-for-exception-as-string> | Dictionary |
| NSExceptionMinimumTLSVersion | String |
| NSExceptionRequiresForwardSecrecy | Boolean |
| NSExceptionAllowsInsecureHTTPLoads | Boolean |
| NSIncludesSubdomains | Boolean |
| NSThirdPartyExceptionMinimumTLSVersion | String |
| NSThirdPartyExceptionRequiresForwardSecrecy | Boolean |
| NSThirdPartyExceptionAllowsInsecureHTTPLoads | Boolean |

# Fragen?

# Vielen Dank

**Kontakt:**
Klaus Rodewig
Appnö UG (haftungsbeschränkt)
Bilker Allee 217
40215 Düsseldorf
kmr@appnoe.de



# Macoun