Practical Security

Rob Napier

github.com/rnapier/practical-security

saltmarch MEDIA

MODS

MOBILE & DISRUPTIVE TECHNOLOGY SUMMIT

5-7 October 2016. Bangalore, India

www.modsummit.com

Encrypt Your Traffic

HTTPS

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

App Transport Security

Leave it alone

Certificate Pinning

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, ...

It’s always riskier to trust yourself and someone else, than to just trust yourself.

Certificate Pinning

```
try! validator = CertificateValidator(
                        certificateURL: certificateURL)

session = URLSession(configuration: .default,
                        delegate: validator,
                        delegateQueue: nil)

task = session.dataTask(with: URLRequest(url: fetchURL)) {
    // ...
}
```

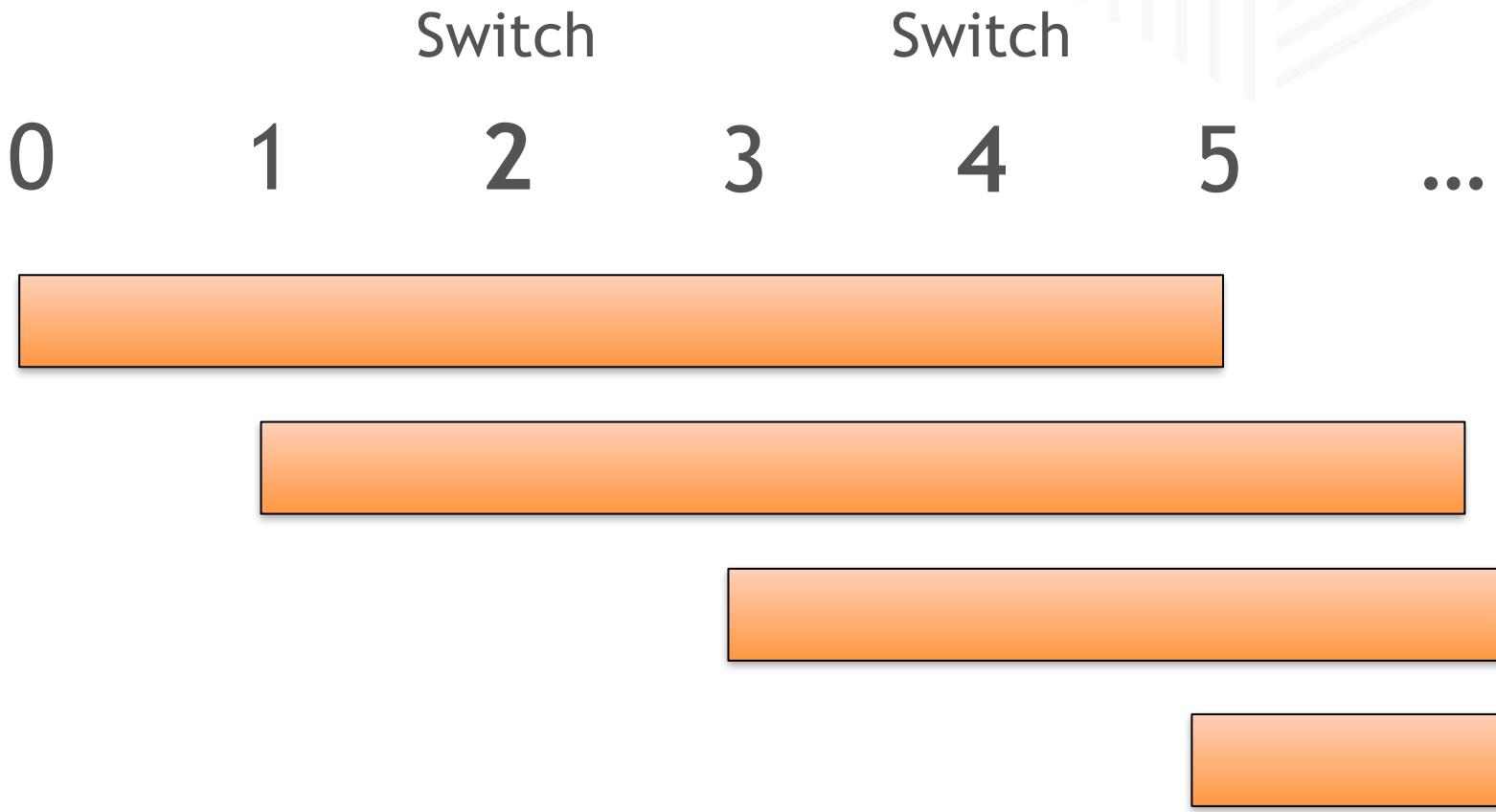
https://github.com/rnapier/CertificateValidator

https://github.com/rnapier/CertificateValidator

https://github.com/datatheorem/TrustKit

https://talk.objc.io/episodes/S01E57-certificate-pinning

Certificate Rotation



Encrypt Your Traffic

» Use HTTPS for all traffic
» Pin your certs
» Rotate certs regularly

https://github.com/rnapier/CertificateValidator

https://github.com/datatheorem/TrustKit

https://talk.objc.io/episodes/S01E57-certificate-pinning

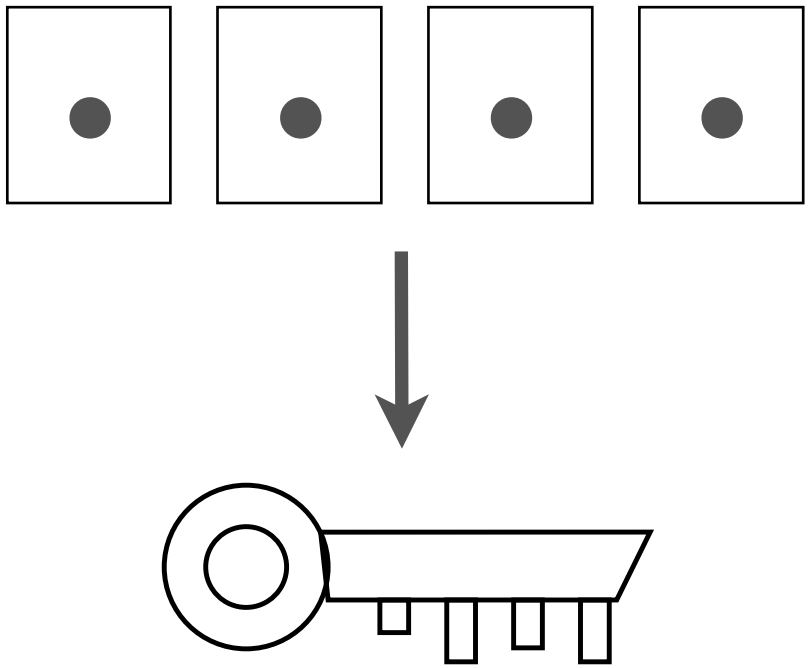
Data Protection

iOS Encryption

Data Protection

Device Encryption

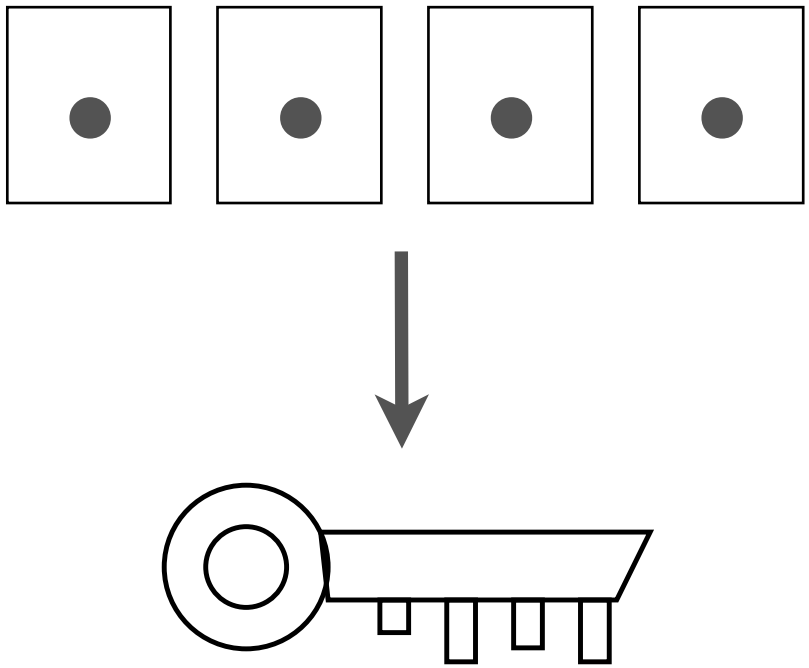
Data Protection (Simplified)



I can see
SSBjYW4g
by my
c2VlIGJ5l
watch,
G15lHdhd
without
GNoLCB3a
taking my
XRob3V0l
hand from
HRha2luZ
the left
yB.teSBoY
grip of
W5kIGZ
the cycle,

`NSFileProtectionComplete`

Data Protection (Simplified)



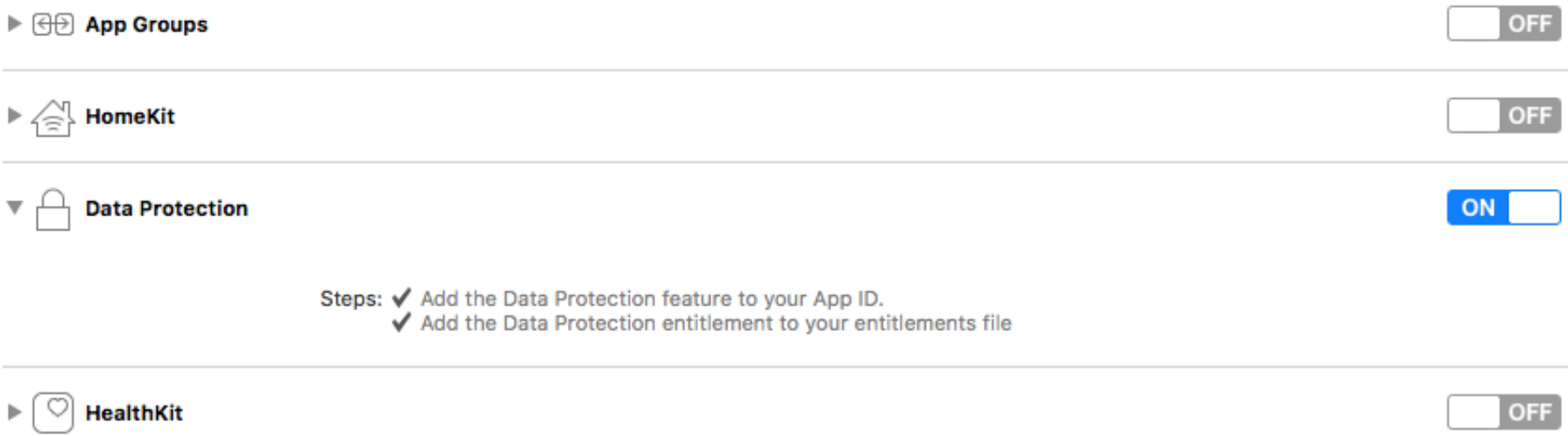
I can see
SSBjYW4g
by my
c2VlIGJ5I
watch,
G15lHdhd
without
GNoLCB3a
taking my
XRob3V0I
hand from
HRha2luZ
the left
yBteSBoY
grip of
W5kIGZ
the cycle,

NSFileProtectionComplete

Protection Levels

» Complete
» Complete Unless Open
» Complete Until First User Authentication

How Easy?


App Groups
OFF
HomeKit
OFF
Data Protection
ON
Steps: ✔ Add the Data Protection feature to your App ID.
✔ Add the Data Protection entitlement to your entitlements file
HealthKit
OFF

Data Protection In Code

```
try data.write(to: url,
               options: .completeFileProtectionUnlessOpen)
```

```
extension FileManager {
    func protectFileAtPath(path: String) throws {
        try setAttributes([
            .protectionKey:
                FileProtectionType.completeUnlessOpen
            ],
                              ofItemAtPath: path)
    }
}
```

UIApplicationDelegate Methods

```
applicationProtectedDataWillBecomeUnavailable(UIApplication)
applicationProtectedDataDidBecomeAvailable(UIApplication)
```

UIApplication Notifications

```
UIApplicationProtectedDataWillBecomeUnavailable
UIApplicationProtectedDataDidBecomeAvailable
```

UIApplication Methods

```
var isProtectedDataAvailable: Bool { get }
```

https://www.apple.com/business/docs/iOS_Security_Guide.pdf

Data Protection

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

https://www.apple.com/business/docs/iOS_Security_Guide.pdf

Handling Passwords

Hashing

Password		Hash
`S3kr3t!`		`d39ee8e54a...`

Choose Your Hash

SHA-2

SHA-224
SHA-256
SHA-384
SHA-512
SHA-512/224
SHA-512/256

Choose Your Hash

SHA-2

SHA-224

SHA-256

SHA-384

SHA-512

SHA-512/224

SHA-512/256

What went wrong At LinkedIn?

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

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 CryptoSwift

let password = Array(“s33krit".utf8)

let salt = Array(
    "com.example.MyGreatSite:robnapier@gmail.com".utf8
)

let bytes = try PKCS5.PBKDF2(password: password,
                             salt: salt,
                             iterations: 10000,
                             variant: .sha256).calculate()

let data = Data(bytes: bytes)
```

https://github.com/krzyzanowskim/CryptoSwift

Store a Hash

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

Good Password Handling

» Hash to hide the password
» Salt to make your hashes unique
» Stretch to make guessing slow
» Hash once more before storing

Correct AES Encryption

Use My Library

https://github.com/RNCryptor

Using RNCryptor

```
// Encryption
let data: NSData = ...
let password = "Secret password"
let ciphertext = RNCryptor.encryptData(data,
                                       password: password)
```

```
// Decryption
do {
    let originalData = try RNCryptor.decryptData(ciphertext,
                                                 password: password)
    // ...
} catch { . . . }
```

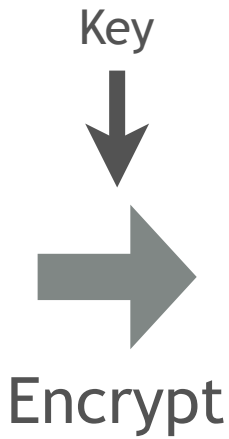
RNCryptor Ports

- » Swift
- » Objective-C
- » ANSI C
- » C++
- » C#
- » Erlang
- » Go
- » Haskell
- » Java
- » PHP
- » Python
- » JavaScript
- » Ruby

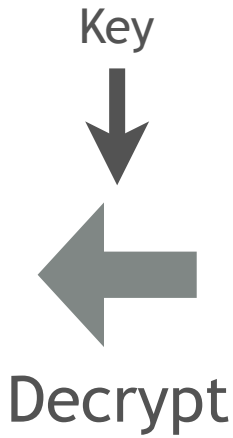
What is Correct AES?

Hold that thought…

P1	P2	P3	P4
P5	P6	P7	P8
P9	P1	P1	P1
P1	P1	P1	P1



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	P1	P1	P1
P1	P1	P1	P1

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

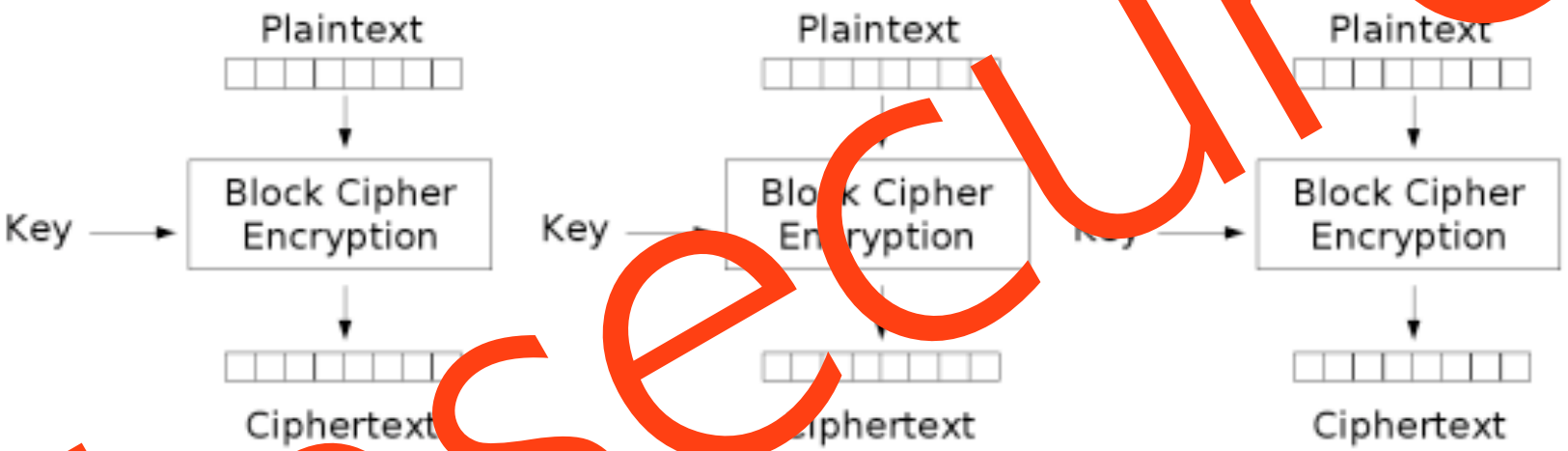
» Best case is ~ 0.00001% of a 128-bit key.

Use a PBKDF
(scrypt, bcrypt, PBKDF2)

To be a secure password-based format, we need a salt for PBKDF2. Ideally it should be totally random.

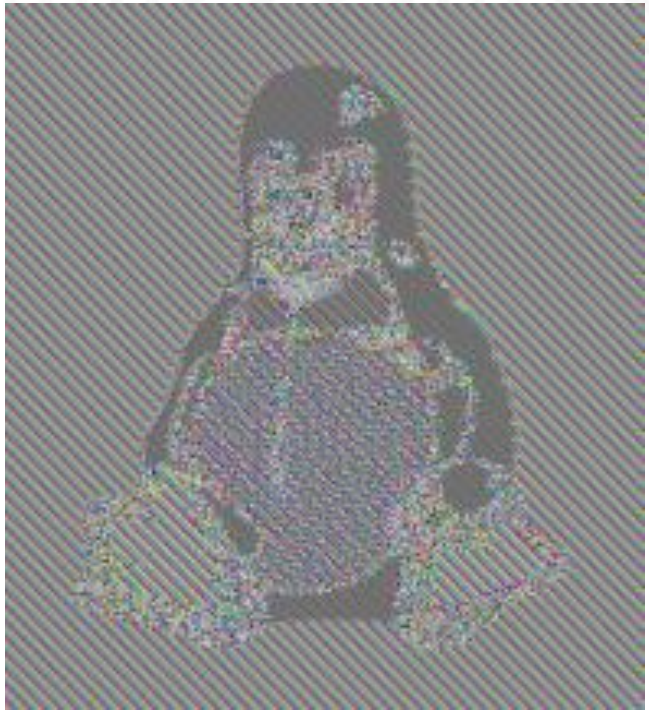
Initialization Vector

And Modes of Operation

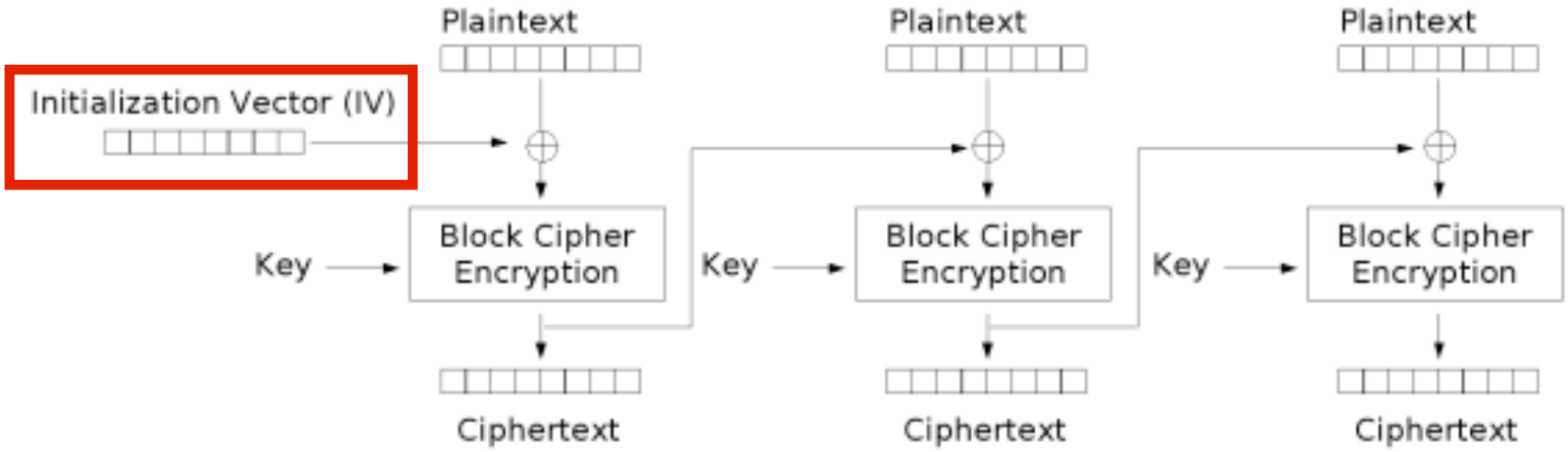


Electronic Codebook (ECB) mode encryption

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

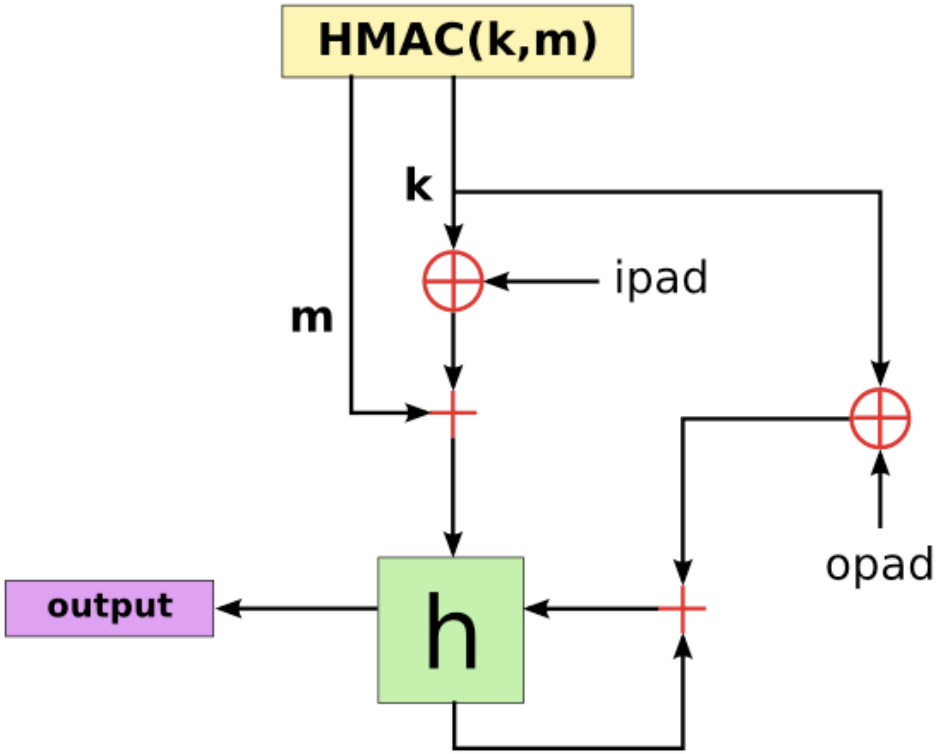
```
CCCryptorStatus CCCryptorCreate(
  CCOperation op,               /* kCCEncrypt, etc. */
  CCAlgorithm alg,              /* kCCAlgorithmDES, etc. */
  CCOptions options,            /* kCCOptionPKCS7Padding, etc. */
  const void *key,              /* raw key material */
  size_t keyLength,
  const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

Use an unpredictable IV, not NULL.

To be a secure format with CBC, we need a random IV.

Unauthenticated Encryption

Hash Based Message Authentication Code



To be a secure format using CBC, we need an HMAC.

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

AES Best Practice

- Key-Derivation Function (PBKDF2)
- Proper Mode (usually CBC)
- Random Initialization Vector
- Authentication (HMAC or authenticated mode)

Practical Security

- Encrypt your traffic with SSL
- Pin and verify your certs (CertificateValidator)
- Encrypt your files with ProtectionComplete
- Salt and stretch your passwords
- Use AES securely

github.com/rnapier/practical-security

robnapier@gmail.com

@cocoaphony

robnapier.net